AD-A040 207 ARMY ARMAMENT RESEARCH AND DEVELOPMENT COMMAND DOVER--ETC F/G 9/2
KSPLIB, CHARACTER STRING PROCESSING LIBRARY ROUTINES, USER MANU--ETC(U)
APR 77 B D CARBREY

UNCLASSIFIED MISD-UM-77-2 NL

1 OF 1
ADA
040207

END

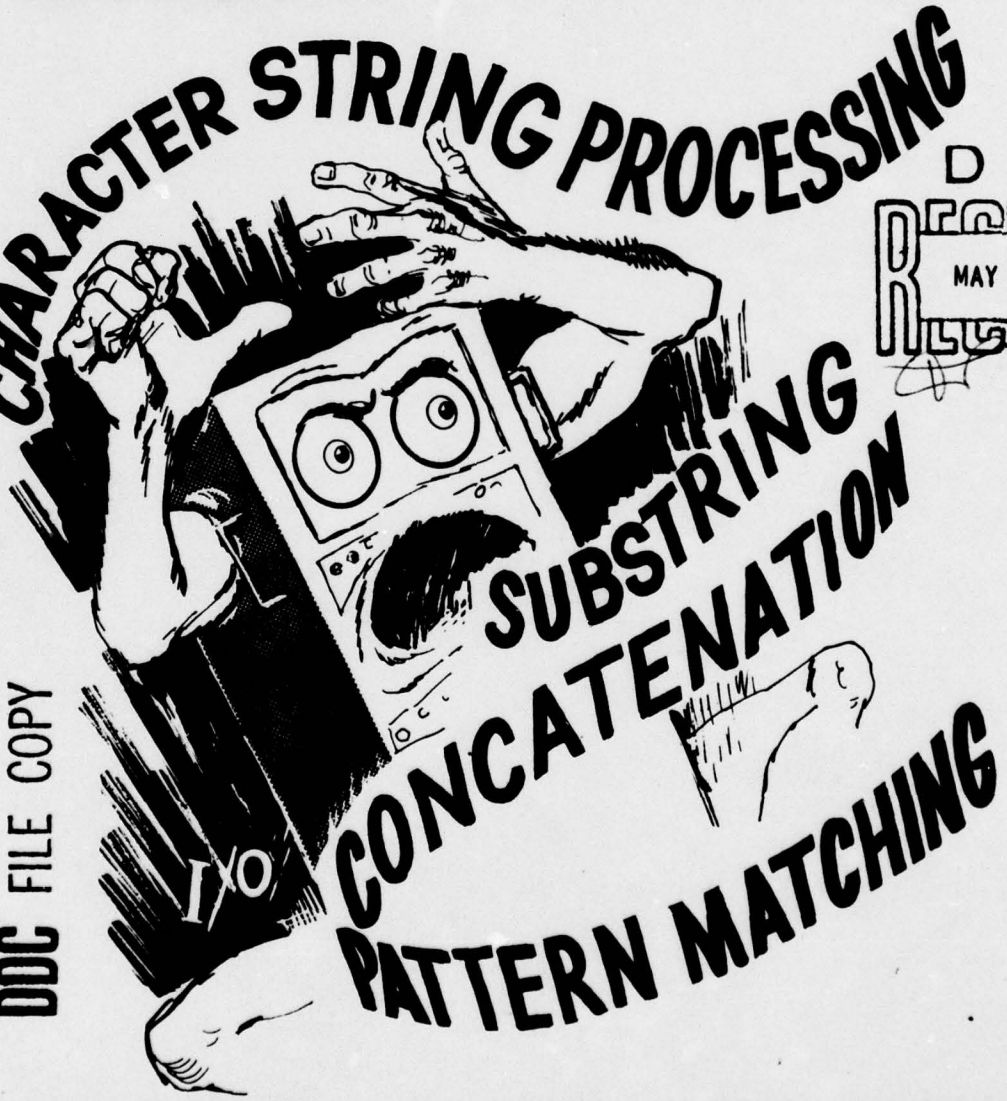
DATE
FILMED
6-77

MANAGEMENT INFORMATION SYSTEMS DIRECTORATE

MISD UM 77-2

KSPLIB
USER MANUAL

AD A040207



DDC
RECEIVED
MAY 24 1977
D

AD No.
DDC FILE COPY

ARRADCOM DOVER, NEW JERSEY

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISCLAIMER

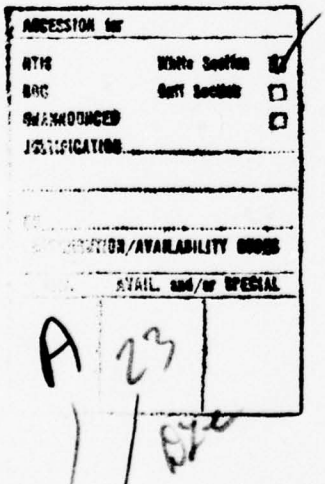
THIS REPORT IS ISSUED FOR INFORMATION AND DOCUMENTATION PURPOSES ONLY. NO REPRESENTATION IS MADE BY THE GOVERNMENT WITH RESPECT TO THE INFORMATION IN THIS REPORT AND THE GOVERNMENT ASSUMES NO LIABILITY OR OBLIGATION WITH RESPECT THERETO: NOR IS THE MATERIAL PRESENTED TO BE CONSTRUED AS THE OFFICIAL POSITION OF THE DEPARTMENT OF THE ARMY, UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

MISD USER MANUAL NO. 77-2

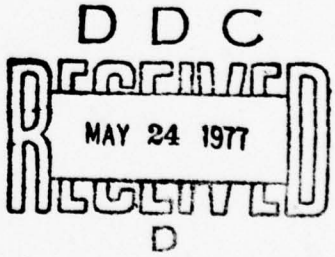
KSPLIB

CHARACTER STRING PROCESSING LIBRARY ROUTINES
USER MANUAL

BRUCE D. CARBREY



APRIL 1977



1473 in rear

MANAGEMENT INFORMATION SYSTEMS DIRECTORATE
ARRADCOM
DOVER, NEW JERSEY 07801

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

ABSTRACT

THIS REPORT DESCRIBES A SET OF GENERAL-PURPOSE, FORTRAN-CALLABLE FUNCTIONS AND SUBROUTINES FOR MANIPULATING CHARACTER-STRINGS. CHARACTER-STRINGS OF ARBITRARY LENGTH CAN BE TREATED AS SIMPLE (NON-DIMENSIONED) VARIABLES. FUNCTIONS ARE PROVIDED FOR STRING DEFINITION, STRING CONCATENATION, SUBSTRING EXTRACTION, STRING COMPARISON, PATTERN MATCHING AND SEARCHING, NUMERIC-STRING/STRING-NUMERIC CONVERSIONS, STRING INPUT/OUTPUT, AND OTHER TASKS. THE ROUTINES DESCRIBED ARE MODERATELY SYSTEM-DEPENDENT, AND ARE CURRENTLY AVAILABLE AS AN OBJECT PROGRAM LIBRARY ON THE CDC 6500/6600 COMPUTERS AT THE ARRADCOM DOVER, NEW JERSEY SITE.

ASSUMPTIONS

THE READER IS EXPECTED TO BE AN EXPERIENCED FORTRAN PROGRAMMER WITH A SOLID UNDERSTANDING OF COMPUTER CONCEPTS AND A GENERAL KNOWLEGE OF THE CDC SCOPE OPERATING SYSTEM AND CDC FORTRAN EXTENDED LANGUAGE.

WARNING

THE FUNCTIONS AND SUBROUTINES DESCRIBED IN THIS REPORT ARE MODERATELY SYSTEM-DEPENDENT. USERS ANTICIPATING A REQUIREMENT FOR TRANSPORTABILITY OF PROGRAMS TO A COMPUTER ENVIRONMENT OTHER THAN A CDC 6X00 COMPUTER WITH SCOPE OPERATING SYSTEM SHOULD SEEK ALTERNATIVE METHODS OR CONSULT THE AUTHOR FOR IMPLEMENTATION GUIDANCE ON OTHER COMPUTERS.

TABLE OF CONTENTS

INTRODUCTION.....1
NOTATION USED IN THIS MANUAL.....8
GENERAL RULES FOR USING THE ROUTINES.....9
DESCRIPTION OF THE ROUTINES.....11

APPENDICIES

A. CONTROL CARDS / COMMANDS.....43
B. SAMPLE PROGRAMS.....44
C. MEMORY REQUIREMENTS FOR ROUTINES.....64

REFERENCES.....65
QUICK REFERENCE INDEX TO ROUTINES.....66

INTRODUCTION

ONE OF THE MOST SERIOUS SHORTCOMINGS OF THE FORTRAN LANGUAGE IN COMPARISON TO MODERN HIGH-LEVEL LANGUAGES SUCH AS ALGOL, PL/1 OR SNOBOL IS THE ABSENCE OF A CHARACTER DATA TYPE. IN MANY LANGUAGES CHARACTER STRINGS OF ANY LENGTH MAY BE DEFINED AND MANIPULATED AS A SIMPLE VARIABLE, JUST AS IS THE CASE FOR REAL OR INTEGER NUMBERS. FORTRAN, HOWEVER, HAS NO SUCH FACILITY, MAKING IT EXTREMELY DIFFICULT TO PROGRAM MANY APPLICATIONS IN WHICH CHARACTER STRINGS MUST BE MANIPULATED (THE FUTURE ANSI FORTRAN PROPOSED BY COMMITTEE X3J3 INCLUDES A LIMITED IMPLEMENTATION OF THE CHARACTER TYPE VARIABLE. FOR BACKGROUND INFORMATION ON STRING HANDLING IN VARIOUS LANGUAGES SEE REFERENCES [6],[7], AND [8]).

THE ROUTINES DESCRIBED IN THIS REPORT GO A LONG WAY TOWARD THE GOAL OF GIVING THE USER A CHARACTER-TYPE VARIABLE AND CHARACTER STRING OPERATORS AS IMPLEMENTED IN THE NEWER HIGH-LEVEL LANGUAGES. THE MOST SIGNIFICANT FEATURE OF THESE ROUTINES IS THAT THEY ENABLE THE USER TO DEFINE AND MANIPULATE CHARACTER STRINGS OF ANY LENGTH IN A MANNER SIMILAR TO OTHER VARIABLES. WITH THESE ROUTINES, ANY LENGTH STRING CAN BE REPRESENTED AS A SIMPLE INTEGER VARIABLE. VARIOUS FUNCTIONS AND SUBROUTINES ARE PROVIDED FOR DEFINING STRINGS, INPUT AND OUTPUT OF STRINGS, CONCATENATION (JOINING TOGETHER) OF STRINGS, SUBSTRING EXTRACTION, COMPARISON OF STRINGS, PATTERN MATCHING, CONVERSION OF STRINGS TO OTHER DATA TYPES, AND OTHER USEFUL TASKS. IN MANY APPLICATIONS, JUDICIOUS USE OF THESE ROUTINES CAN REDUCE THE NUMBER OF LINES OF CODE REQUIRED FOR A PROGRAM BY A FACTOR OF 10 OR BETTER, AND REDUCE DEVELOPMENT TIME BY EVEN LARGER FACTORS.

AS AN EXAMPLE OF THE USEFULNESS OF THE CHARACTER STRING ROUTINES PRESENTED HERE, CONSIDER THIS HYPOTHETICAL ANECDOTE...

PROGRAMMER ARA MODWUN HAS A PROBLEM TO CODE IN FORTRAN. THE PROGRAM REQUIREMENTS INCLUDE THE NEED TO READ A PROJECT NAME FROM A CARD AND PRINT IT OUT IN THE MIDDLE OF A SENTENCE ON THE LINE PRINTER, SO THAT THE RESULTING LINE WILL READ,

THE <NAME> IS UNDER DEVELOPMENT.

WHERE <NAME> IS THE TEXT READ FROM THE CARD. FOR INSTANCE, IF THE CARD READ, " XM650E3 ROCKET ASSISTED PROJECTILE", THEN THE PRINTOUT SHOULD READ,

THE XM650E3 ROCKET ASSISTED PROJECTILE IS UNDER DEVELOPMENT.

ON THE OTHER HAND, IF THE CARD READ, " M1 RIFLE", THEN THE LINE PRINTED SHOULD READ,

THE M1 RIFLE IS UNDER DEVELOPMENT.

THIS PROBLEM REQUIRES ARA TO PERFORM A SIMPLE CHARACTER STRING MANIPULATION. IN FORTRAN, THOUGH, THE PROBLEM IS NOT ALL THAT TRIVIAL. ONE OF THE THINGS THAT MAKES THE PROBLEM NON-TRIVIAL IS THE FACT THAT THE NAME READ IS OF ARBITRARY LENGTH, SO HE CANT ALLOCATE A FIXED LENGTH OF SPACE FOR THE NAME IN THE FORMAT STATEMENT WHEN PRINTING THE SENTENCE, SINCE ARA MIGHT WIND UP WITH HIS SENTENCE LOOKING LIKE,

THE MI RIFLE IS UNDER DEVELOPMENT.

WHICH LOOKS SLOPPY AND IS DIFFICULT TO READ.
TO SOLVE THIS PROBLEM, ARA CODED THE FOLLOWING PROGRAM SEGMENT.

```
      ...
      DIMENSION NAME(80)
      DIMENSION IPRED(22)
C        PRE-DEFINE PREDICATE OF SENTENCE.
      DATA IPRED / 1H , 1HI, 1HS, 1H ,
     X 1HU, 1HN, 1HD, 1HE, 1HR, 1H ,
     X 1HD, 1HE, 1HV, 1HE, 1HL, 1HO, 1HP, 1HM, 1HE, 1HN, 1HT, 1H. /
C        READ NAME FORM CARD.
      READ(5,105)(NAME(L),L=1,80)
  105 FORMAT(80A1)
C        COUNT BACKWARDS TO CUT OFF ALL TRAILING BLANKS FROM NAME
      DO 100 I=1,79
      J=81-I
      IF(NAME(J).NE.1H )GO TO 101
  100 CONTINUE
  101 CONTINUE
      WRITE(6,106)(NAME(L),L=1,J),(IPRED(L),L=1,22)
  106 FORMAT(* THE*,1 2A1)
      ...
```

THE ABOVE PROGRAM WORKS, BUT ILLUSTRATES HOW A LOT OF CODE MAY BE REQUIRED FOR EVEN THE MOST TRIVIAL TEXT MANIPULATION. IF ARA HAD KNOWN HOW TO USE THE ROUTINES IN THIS MANUAL, HE COULD HAVE REPLACED THE ENTIRE BLOCK OF CODE ABOVE WITH A SINGLE STATEMENT (INCLUDING READING AND PRINTING), BY SIMPLY CODING,

```
      CALL KOUT(KON(" THE"),KIN(5),KON(" IS UNDER DEVELOPMENT.")))
```

THIS ONE STATEMENT READS THE NAME FROM A CARD AND PRINTS THE DESIRED OUTPUT ON THE PRINTER. ALTHOUGH THE STATEMENT MAY LOOK COMPLEX, IT IS ACTUALLY QUITE EASY TO CONSTRUCT, ONCE YOU HAVE MASTERED THE BASIC FUNCTIONS PRESENTED IN THIS MANUAL.

FUNDAMENTAL CHARACTER STRING OPERATIONS

THIS SECTION INTRODUCES SOME OF THE BASIC OPERATIONS INVOLVED WITH CHARACTER STRINGS AS THEY APPLY TO THIS REPORT. LATER WE SHALL EXAMINE THE FUNCTIONS AND SUBROUTINES IN DETAIL.

1. STRING DEFINITION.

A CHARACTER STRING IS A GROUP OF FROM 0 TO 500 CHARACTERS TREATED AS A SINGLE ENTITY. A CHARACTER STRING HAVING 0 CHARACTERS IS CALLED THE NULL STRING.

TO DEFINE A CHARACTER STRING CONSTANT, WE CAN USE FUNCTION KON. FOR EXAMPLE,

```
ISTR=KON("ARRADCOM")
JSTR=KON("DOVER, NEW JERSEY 07801")
```

THE ABOVE TWO STATEMENTS DEFINE TWO STRINGS. THE FIRST STRING HAS 8 CHARACTERS. THE VALUE RETURNED BY FUNCTION KON IS CALLED THE **STRING REPRESENTATION.** IN THE FIRST CASE THE STRING REPRESENTATION OF "ARRADCOM" WAS ASSIGNED TO ISTR. STRING REPRESENTATIONS ARE ALWAYS ASSIGNED TO **INTEGER** VARIABLES. ANY STRING MAY BE REPRESENTED BY A SIMPLE INTEGER VARIABLE. THE SECOND STATEMENT GENERATES ANOTHER STRING CONSTANT REPRESENTATION ASSIGNED TO JSTR. FOR A STRING OF 23 CHARACTERS.

FOR THE PRESENT WE NEED NOT BE CONCERNED WITH WHAT THE INTERNAL FORM OF THE REPRESENTATION OF THE STRING IS; IT IS SUFFICIENT TO KNOW THAT ISTR AND JSTR UNIQUELY DEFINE THE STRINGS.

2. STRING CONCATENATION

CONCATENATION MEANS "JOINING TOGETHER". TWO OR MORE STRINGS MAY BE CONCATENATED BY USING FUNCTION KAT. FOR EXAMPLE, THE STRINGS PREVIOUSLY DEFINED CAN BE JOINED BY,

```
KSTR=KAT(ISTR,JSTR)
```

THE VARIABLE KSTR NOW CONTAINS THE REPRESENTATION OF "ARRADCOMDOVER, NEW JERSEY 07801".

3. SUBSTRING EXTRACTION.

SUBSTRING IS THE INVERSE FUNCTION OF CONCATENATION. SUBSTRING MEANS TO TAKE A STRING APART. IN ORDER TO EXTRACT A PORTION OF A LARGER CHARACTER STRING, FUNCTION KSUB MAY BE USED. FOR EXAMPLE,

```
      NEWSTR=KSUB(ISTR,4,5)
```

NEWSTR IS DEFINED AS THE STRING REPRESENTATION OF THE STRING "ADCOM". BECAUSE KSUB EXTRACTED 5 CHARACTERS BEGINNING AT THE FOURTH CHARACTER OF ISTR. SIMILARLY, KSUB(JSTR,8,3) RESULTS IN THE STRING REPRESENTATION OF "NEW".

4. STRING INPUT/OUTPUT.

TWO ROUTINES ARE PROVIDED FOR GENERAL-PURPOSE INPUT AND OUTPUT OF STRINGS. FUNCTION KIN MAY BE USED LIKE A READ STATEMENT. FOR EXAMPLE,

```
      LSTR=KIN(5)
```

THIS STATEMENT READS A RECORD (CARD) FROM UNIT 5 AND ASSIGNS THE STRING REPRESENTATION OF THE CARD IMAGE TO LSTR. IF THE CARD READ FROM TAPE5 CONSISTED OF,

```
THIS IS AN EXAMPLE OF A CHARACTER STRING
```

THEN LSTR WOULD BE THE REPRESENTATION OF THE 40 CHARACTER STRING, "THIS IS AN EXAMPLE OF A CHARACTER STRING". LATER A NUMBER OF OPTIONS WILL BE EXPLAINED FOR FUNCTION KIN.

TO OUTPUT A STRING, SUBROUTINE KOUT MAY BE USED. FOR EXAMPLE TO OUTPUT THE STRING PREVIOUSLY ASSIGNED TO ISTR, WE COULD SIMPLY CODE.

```
      CALL KOUT(6,ISTR)
```

THIS WOULD OUTPUT THE DESIRED STRING TO UNIT 6. IF UNIT 6 WERE A PRINTER, THEN THE STRING "RRADCOM" WOULD BE PRINTED (THE FIRST CHARACTER WOULD BE LOST AS CARRIAGE CONTROL, JUST LIKE ANY OTHER WRITE STATEMENT). SEVERAL CARRIAGE CONTROL OPTIONS MAY BE SPECIFIED IN THE CALL TO KOUT TO AVOID LOSS OF THE FIRST CHARACTER AS CARRIAGE CONTROL, IF DESIRED. THESE WILL BE INTRODUCED LATER.

ALREADY WE CAN SEE THAT IT WOULD BE VERY EASY TO WRITE A SECTION OF CODE TO COPY A CARD FROM THE READER TO THE PRINTER. WE COULD SIMPLY USE THE STATEMENT,

```
      CALL KOUT(6,KIN(5))
```

5. STRING COMPARISON.

USING REGULAR FORTRAN STATEMENTS, IT IS EASY TO COMPARE NUMBERS BY USING AN IF STATEMENT. FOR INSTANCE,

```
IF(XVAL .GE. YVAL) GO TO 23
```

COMPARES XVAL AND YVAL. TWO STRINGS MAY BE COMPARED IN A SIMILAR MANNER USING FUNCTION LEXIC. LEXIC COMPARES TWO STRINGS FOR LEXICAL SEQUENCE (THAT IS, FOR DICTIONARY ORDER). FOR EXAMPLE,

```
IF( LEXIC( ISTR, "GE", JSTR )) GO TO 23
```

COMPARES STRING REPRESENTATIONS ISTR AND JSTR. IF THE STRING REPRESENTED BY ISTR WOULD FALL AFTER JSTR IN THE DICTIONARY, OR IF BOTH STRINGS ARE IDENTICAL, THEN CONTROL IS TRANSFERRED TO STATEMENT 23. OTHERWISE, CONTROL PROCEEDS TO THE NEXT STATEMENT AFTER THE IF STATEMENT. FUNCTION LEXIC IS A LOGICAL FUNCTION AND ALWAYS RETURNS EITHER .TRUE. OR .FALSE.. THE SECOND ARGUMENT PASSED TO LEXIC IS THE KEYWORD DESCRIBING THE TYPE OF COMPARISON DESIRED, AND MAY BE ANY OF THE FOLLOWING.

"LT" "LE" "EQ" "NE" "GE" "GT"

6. STRING LENGTH.

THE LENGTH OF A STRING IS THE NUMBER OF CHARACTERS IT CONTAINS. TO FIND THE LENGTH OF ANY STRING, USE FUNCTION LEN. FOR EXAMPLE,

```
NCHAR=LEN(ISTR)
```

WILL ASSIGN NCHAR = 8, BASED ON OUR PREVIOUS DEFINITION OF ISTR.

7. STRING PATTERN MATCHING.

PATTERN MATCHING INVOLVES SEARCHING A STRING FOR ANY SUBSTRING WHICH MATCHES A GIVEN STRING. FOR EXAMPLE, IF A STRING CONSISTS OF "THIS IS A STRING" AND THE PATTERN IS "S IS", THEN WE CAN SAY THE PATTERN MATCHES THE STRING STARTING AT THE 4TH CHARACTER. THE PATTERN MATCHING FUNCTION IS FUNCTION MATCH, AND IT RETURNS THE COLUMN OF THE STRING AT WHICH THE GIVEN PATTERN MATCHES. FOR EXAMPLE,

```
IPAT = KON(" ")
IBLANK=MATCH(JSTR,"FIRST",IPAT)
```

THIS STATEMENT RESULTS IN IBLANK=7 BASED ON OUR PRIOR DEFINITIONS, BECAUSE MATCH SEARCHES FOR THE FIRST OCCURRANCE OF A BLANK (THE PATTERN) IN THE STRING REPRESENTED BY JSTR. THE SECOND ARGUMENT IS A KEYWORD TELLING MATCH WHAT KIND OF

A SEARCH TO MAKE. FOR INSTANCE, IF WE HAD USED "FIRSTNOT" INSTEAD OF "FIRST", MATCH WOULD HAVE RETURNED THE POSITION OF THE FIRST NON-BLANK CHARACTER (WHICH WOULD BE 1 IN THE EXAMPLE). LATER WE SHALL SEE HOW OTHER KEYWORDS MAY BE USED AND HOW MORE THAN ONE PATTERN MAY BE SEARCHED FOR SIMULTANEOUSLY. PATTERN MATCHING IS OFTEN USED IN CONJUNCTION WITH SUBSTRING TO BREAK UP A STRING INTO ITS COMPONENTS. FOR EXAMPLE, IF WE WISHED TO EXTRACT AS A SUBSTRING ALL THE INFORMATION CONTAINED IN PARENTHESES IN ANOTHER STRING, WE COULD USE THE FOLLOWING STATEMENTS.

```
C          ASSUME INSTR IS STRING TO BE SEARCHED...
C          LOCATE COLUMN CONTAINING 1ST "("...

        ICOL1=MATCH(INSTR,"FIRST",KON( "(" ))

C          ...AND COLUMN CONTAINING LAST ")" ...

        ICOLL=MATCH(INSTR,"LAST",KON( ")" ))

C          TRANSFER  STRING BETWEEN PARENTHESES TO IPSTR...

        IPSTR=KSUB(INSTR,ICOL1+1,ICOLL-ICOL1+1)
```

8. **STRING CONVERSION.**

THREE FUNCTIONS ARE PROVIDED FOR CONVERSION OF STRINGS TO NUMBERS AND NUMBERS TO STRINGS. FUNCTION INTVAL RETURNS THE INTEGER VALUE OF THE NUMBER REPRESENTED BY THE STRING. FOR EXAMPLE,

```
        NUMSTR=KON(" -237 ")
        N=INTVAL(NUMSTR)
        NSQUAR = N*N
```

IN THIS CASE, THE VALUE OF N WILL BE -237. SIMILARLY, FUNCTION RLVAL CAN BE USED FOR STRING-TO-REAL CONVERSIONS. AS A SIMPLE EXAMPLE OF THE UTILITY OF THESE FUNCTIONS, SUPPOSE WE WANTED TO READ A DATA CARD WHICH CONTAINED A NUMBER IN PARENTHESES SOMEWHERE ON THE CARD. IF WE WISHED TO USE THE NUMERIC VALUE INSIDE THOSE PARENTHESES FOR A COMPUTATION. WE COULD DO SO BY FIRST READING THE CARD USING FUNCTION KIN: SECOND, EXTRACTING THE SUBSTRING BETWEEN THE PARENTHESES, AS ALREADY ILLUSTRATED FOR FUNCTION MATCH: FINALLY, CODING,

```
NUM=INTVAL(IPSTR)
```

AT THIS POINT NUM IS THE INTEGER VALUE OF THE NUMBER WE WANT, AND CAN BE DIRECTLY USED IN COMPUTATION.

FUNCTION KSR PROVIDES THE INVERSE FUNCTION OF BOTH INTVAL AND RLVAL. IT CONVERTS A REAL OR INTEGER NUMBER TO A STRING REPRESENTATION OF THE NUMBER. FOR EXAMPLE,

```
NUM=23+14
NSTR=KSR(NUM)
CALL KOUT(6,KON(" NUM="),NSTR)
```

WILL RESULT IN THE PRINTING OF,

NUM=37

FUNCTION KSR HAS SEVERAL OPTIONAL FORMS WHICH WE SHALL EXAMINE LATER. IT IS IMPORTANT TO NOTE THOUGH THAT KSR IS MUCH MORE POWERFUL THAN FORTRAN WRITE/FORMAT STATEMENTS BECAUSE THE SIZE OF THE STRING CAN BE ADJUSTED AUTOMATICALLY TO THE MAGNITUDE OF THE NUMBER. FOR EXAMPLE, KSR (0) WILL GIVE A 1 CHARACTER STRING, "0", BUT KSR (-23.4588655) WILL PRODUCE THE STRING REPRESENTATION OF "-23.4588655". WHICH IS 11 CHARACTERS. THIS CAPABILITY IS CALLED DYNAMIC FORMATTING AND IS VERY USEFUL IN MANY APPLICATIONS.

9. STRING ROUTINE INITIALIZATION.

BEFORE ANY OF THE FUNCTIONS OR SUBROUTINES IN THIS MANUAL MAY BE USED, SUBROUTINE KINIT MUST BE CALLED. FOR THE PRESENT, WE SHALL LIMIT OUR DISCUSSION OF KINIT TO SAY THAT IT HAS THE FORM,

```
CALL KINIT(0)
```

LATER OTHER ARGUMENTS WILL BE INTRODUCED.
KINIT DOES NOT PERFORM ANY FUNCTION THAT IS APPARENT TO THE USER BUT IS ESSENTIAL TO THE OPERATION OF THE ROUTINES. FAILURE TO CALL KINIT WILL RESULT IN THE FATAL DAYFILE DIAGNOSTIC.

STOP KINIT NEVER CALLED.

THESE ARE THE BASIC CONCEPTS AND FUNCTIONS INVOLVED WITH THE HANDLING OF STRINGS. A NUMBER OF OTHER FUNCTIONS AND SUBROUTINES WILL BE INTRODUCED LATER, AND A NUMBER OF OPTIONAL FORMS WILL BE PRESENTED FOR THE ROUTINES ALREADY INTRODUCED. SO AS TO IMPROVE THE UTILITY OF THE CHARACTER STRING HANDLING PACKAGE.

NOTATION USED IN THIS MANUAL

THE FOLLOWING NOTATION IS USED IN DESCRIBING THE SUBROUTINES AND FUNCTIONS IN THIS MANUAL:

WHEN DESCRIBING ARGUMENT LISTS FOR STRING ROUTINES, IF AN ITEM IS ENCLOSED IN ANGLE BRACKETS, FOR INSTANCE,

<INTEGER>

THEN IT IS A DESCRIPTION OF THE TYPE OF ENTRY REQUIRED. FOR EXAMPLE, IF WE WERE TO DESCRIBE THE SYNTAX OF AN ORDINARY FORTRAN WRITE STATEMENT, WE MIGHT WRITE,

WRITE (<UNIT NO.>,<FORMAT LABEL>) <VARIABLE LIST>

OFTEN ARGUMENTS ARE OPTIONAL, AND MAY BE OMITTED IF DESIRED. AN ITEM CONTAINED IN SQUARE BRACKETS IS OPTIONAL. TO EXPAND OUR EXAMPLE FOR THE FORTRAN WRITE STATEMENT, RECALL THAT THE <FORMAT LABEL> MAY BE OMITTED FOR UNFORMATTED WRITES, AND THAT WE NEED NOT HAVE ANY <VARIABLE LIST>. THEREFORE WE COULD IMPROVE OUR DEFINITION TO READ,

WRITE(<UNIT NO.> [,<FORMAT LABEL>]) [<VARIABLE LIST>]

THREE DOTS (...) IS USED TO INDICATE AN ELEMENT WHICH MAY BE REPEATED AN ARBITRARY NUMBER OF TIMES. WE COULD REPLACE <VARIABLE LIST> IN OUR DEFINITION OF FORTRAN WRITE WITH <VARIABLE>[,<VARIABLE>...] TO SHOW ONE POSSIBLE FORM OF <VARIABLE LIST> AS AN ARBITRAY NUMBER OF VARIABLES SEPARATED BY COMMAS.

GENERAL RULES

IT IS EXTREMELY IMPORTANT THAT VARIABLES USED WITH THE STRING FUNCTIONS AND SUBROUTINES BE OF THE RIGHT TYPE. ALL STRING REPRESENTATIONS **MUST** BE ASSIGNED TO AN **INTEGER** VARIABLE. ASSIGNING A STRING REPRESENTATION TO ANY OTHER TYPE OF VARIABLE WILL RESULT IN THE FATAL DAYFILE DIAGNOSTIC.

STOP ILLEGAL STRING REPRESENTATION

BEING PRODUCED WHEN THE VARIABLE IS SUBSEQUENTLY REFERENCED IN A STRING FUNCTION OR SUBROUTINE. AS A MEMORY AID, REMEMBER THAT ALL FUNCTIONS WHICH RETURN A STRING REPRESENTATION AS THE VALUE OF THE FUNCTION BEGIN WITH THE LETTER K. THE ONLY FUNCTION IN THIS MANUAL RETURNING A REAL RESULT IS RLVAL. THE ONLY FUNCTION RETURNING A LOGICAL VALUE IS LEXIC. ALL OTHER FUNCTIONS RETURN INTEGER VALUES.

THIS RULE IS SO IMPORTANT IT BEARS REPEATING...

ANY <STRING REPRESENTATION> CAN ONLY BE ASSIGNED TO A VARIABLE THAT IS TYPE INTEGER.

FOR ALL ROUTINES IN THIS MANUAL, UNLESS OTHERWISE SPECIFIED, **ALL ARGUMENTS MUST BE TYPE INTEGER.**

ARRAYS OF STRINGS

WE MAY USE ARRAYS OF CHARACTER STRINGS JUST AS WE MAY USE ARRAYS OF NUMERIC VARIABLES. EACH ELEMENT OF THE ARRAY CAN REPRESENT A STRING OF UP TO 500 CHARACTERS. FOR EXAMPLE,

```
      DIMENSION IST(5)
      ...
      DO 100 I=1,5
  100 IST(I)=KIN(5)
```

THIS READS FIVE CARDS AND ASSIGNS EACH CARD-IMAGE-STRING TO A SEPARATE ELEMENT OF ARRAY IST. OF COURSE, IST MUST BE AN INTEGER ARRAY.

PASSING STRINGS TO SUBROUTINES

STRING REPRESENTATIONS MAY BE PASSED TO OTHER ROUTINES JUST LIKE ANY OTHER VARIABLE. THEY MAY APPEAR IN COMMON OR AS ARGUMENTS IN A SUBROUTINE OR FUNCTION CALL. ATTEMPTING TO PERFORM ANY OPERATIONS ON STRING REPRESENTATIONS OTHER THAN THOSE PROVIDED BY ROUTINES IN THIS MANUAL WILL RESULT IN UNDEFINED RESULTS. FOR INSTANCE, TRYING TO PRINT A STRING REPRESENTATION USING AN ORDINARY PRINT OR WRITE STATEMENT WILL

PRODUCE GARBAGE, NOT THE STRING DESIRED. THE ONLY LEGAL ARITHMETIC OPERATIONS WHICH MAY BE PERFORMED ON STRING REPRESENTATIONS ARE,

1. SIMPLE ASSIGNMENT STATEMENTS, FOR EXAMPLE,

```
KSTR=JSTR
```

2. ASSIGNING THE VALUE 0 TO A STRING REPRESENTATION, WHICH DEFINES A NULL STRING. FOR EXAMPLE,

```
KSTR=0
```

SETS KSTR TO THE REPRESENTATION OF A NULL STRING (0 CHARACTERS).

THERE ARE TWO KEYS TO SUCCESS IN USING THESE ROUTINES...

1. REMEMBER TO CALL KINIT BEFORE ANY OTHER STRING ROUTINE.
2. USE THE RIGHT TYPE OF VARIABLE.

KEEPING IN MIND THESE IMPORTANT GENERAL RULES, LET US EXAMINE THE STRING ROUTINES IN MORE DETAIL.

INTEGER FUNCTION KON - DEFINE A STRING CONSTANT.

LEGAL FORMATS...

```
<STRING REP.> = KON(<LITERAL STRING>)                [1.1]
<STRING REP.> = KON(<LITERAL STRING>,<LENGTH>)       [1.2]
<STRING REP.> = KON(<LITERAL STRING>,-<PADDING>)     [1.3]
```

ARGUMENTS...

<LITERAL STRING> IS EITHER A CHARACTER STRING ENCLOSED IN QUOTATION MARKS, OR A HOLLERITH CONSTANT OF THE FORM, <NO.CHAR.>H<TEXT>. UNDER NO CIRCUMSTANCES WHATSOEVER CAN THIS ARGUMENT BE A VARIABLE. (NOTE... ON SOME PRINTERS AND KEYPUNCHES, THE QUOTE (") APPEARS AS A "NOT-EQUAL" SYMBOL. THE CORRECT KEYPUNCH FOR THE QUOTE IS AN 8-4 PUNCH ON AN 026 KEYPUNCH, AND AN 8-7 PUNCH ON AN 029 KEYPUNCH.)

<LENGTH> IS AN OPTIONAL PARAMETER SPECIFYING THE COMPLETE LENGTH DESIRED FOR THE STRING. IT MAY BE A VARIABLE. IF <LENGTH> IS GREATER THAN THE LENGTH OF THE <LITERAL STRING>, ADDITIONAL BLANKS WILL BE ADDED AS REQUIRED. IF <LENGTH> IS SHORTER THAN <ACTUAL STRING>, THE STRING WILL BE TRUNCATED.

<PADDING> IS AN OPTIONAL PARAMETER DISTINGUISHED FROM <LENGTH> BY THE FACT THAT IT IS PRECEEDED BY A MINUS SIGN. <PADDING> IS A NUMBER SPECIFYING THE NUMBER OF BLANKS TO BE APPENDED TO THE END OF THE <LITERAL STRING>.

IMPORTANT NOTES...

DUE TO THE INTERNAL DATA REPRESENTATION OF BLANKS IN CDC FORTRAN EXTENDED, IT IS IMPOSSIBLE TO DISTINGUISH THE NUMBER OF TRAILING BLANKS ON A <LITERAL STRING>. FOR INSTANCE, "HI " AND "HI " BOTH HAVE PRECISELY THE SAME REPRESENTATION IN CDC FORTRAN. FOR THIS REASON, IT IS NECESSARY FOR FUNCTION KON TO IMPLEMENT THE FOLLOWING RULE TO AVOID AMBIGUITY...

WHEN USING FORM [1.1], ANY TRAILING BLANKS IN <LITERAL STRING> WILL BE AUTOMATICALLY REMOVED BY KON. IF <LITERAL STRING> IS COMPOSED SOLELY OF BLANKS, THE RESULT WILL BE A SINGLE BLANK. FROM THIS RULE, IT IS APPARENT THAT KON("HI") AND KON("HI ") BOTH DEFINE THE SAME STRING, NAMELY THE TWO CHARACTER STRING, "HI". IF IT IS DESIRED TO HAVE ONE OR MORE TRAILING BLANKS INCLUDED IN THE STRING, THEN USE FORM [1.2] OR FORM [1.3] INSTEAD. FOR EXAMPLE, KON(" HI",-1) WILL PAD THE <LITERAL STRING> WITH ONE TRAILING BLANK, RESULTING IN " HI ". THE CHARACTERS IN THE <LITERAL STRING> MAY BE ANY CHARACTERS IN THE 64 CHARACTER ASCII SUBSET WITH THE EXCEPTIONS NOTED BELOW.

1. THE COLON (:) MAY NOT BE THE LAST CHARACTER OF A STRING.
2. TWO COLONS MAY NEVER APPEAR TOGETHER. AT LEAST ONE OTHER CHARACTER MUST INTERVENE.
3. THE QUOTE CHARACTER (OR NOT-EQUALS SIGN ON SOME PRINTERS AND PUNCHES) SHOULD NOT BE USED IN THE FORM OF <LITERAL STRING> WHICH IS DELIMITED BY THE QUOTE MARKS. IT MAY BE USED FREELY IN THE HOLLERITH-COUNT FORM. (EXCEPTION... THE CHARACTER " MAY BE SUCCESSFULLY REPRESENTED BY REPLACING IT WITH "" IN A STRING DELIMITED BY " CHARACTERS). THESE RULES STEM FROM THE CDC IMPLEMENTATION OF RECORDS, NOT FROM ANY IDIOSYNCRASIES OF KON (SEE REFERENCES [3],[4], AND [5]).

EXAMPLES OF LEGAL STRING DEFINITIONS...

FUNCTION CALL...	RESULT REPRESENTS...
IA=KON("THIS IS A STRING")	"THIS IS A STRING"
IB=KON(16HTHIS IS A STRING)	"THIS IS A STRING"
IC=KON("SO IS THIS ")	"SO IS THIS"
ID=KON(13HSO IS THIS)	"SO IS THIS"
IE=KON("PADDED STRING",-3)	"PADDED STRING "
IG=KON("PADDED STRING ",-3)	"PADDED STRING "
IH=KON("TRUNCATED STRING",6)	"TRUNCA"
II=KON(" ")	" "
IJ=KON(10HSAY "HI")	"SAY "HI""
IK=KON("SAY ""HI"" ",-1)	"SAY "HI" "
IL=KON("LENGTHENED STRING",20)	"LENGTHENED STRING "

EXAMPLES OF ILLEGAL STRING DEFINITIONS...

FUNCTION CALL	REASON FOR ERROR
DATA LS /3HHI /	
IP=KON(LS)	<LITERAL STRING> MAY NOT BE VAR.
STRING=KON("HELLO")	<STRING REP> MAY NOT BE REAL VAR.
IT=KON("ANSWER:")	COLON MAY NOT BE LAST CHARACTER.

SUBROUTINE KONLST - DEFINE ARRAY OF STRING CONSTANTS

LEGAL FORMAT...

CALL KONLST(<INT. ARRAY>,<LIT. STRING>[,<LIT. STRING>...]) [2.1]

ARGUMENTS...

<INT. ARRAY> IS AN INTEGER ARRAY TO RECEIVE THE STRING REPRESENTATIONS.
<LIT. STRING> IS A LITERAL STRING AS DESCRIBED FOR FUNCTION KON. IT MAY NOT BE A VARIABLE.

ANY NUMBER OF <LITERAL STRING>S MAY BE SPECIFIED, SEPARATED BY COMMAS, AND USING CONTINUATION CARDS IF NEEDED.

NOTE... IF THIS FUNCTION, OR OTHER FUNCTIONS WITH VARIABLE-LENGTH ARGUMENT LISTS, IS CALLED SEVERAL TIMES WITH DIFFERENT NUMBERS OF ARGUMENTS (IN THE SAME PROGRAM), THE FORTRAN COMPILER WILL PRODUCE THE INFORMATIVE DIAGNOSTIC, "ARGUMENT COUNT INCONSISTENT WITH PRIOR USAGE". THIS DIAGNOSTIC SHOULD BE IGNORED.

EXAMPLE 2.1...

```
      DIMENSION ISTRG(5)
      CALL KINIT(9)
      ...
      CALL KONLST(ISTRG,"MOVE","DRAW","DASH","TEXT","ERASE")
```

THE ABOVE EXAMPLE IS FUNCTIONALLY EQUIVALENT TO...

```
      DIMENSION ISTRG(5)
      CALL KINIT(9)
      ...
      ISTRG(1)=KON("MOVE")
      ISTRG(2)=KON("DRAW")
      ISTRG(3)=KON("DASH")
      ISTRG(4)=KON("TEXT")
      ISTRG(5)=KON("ERASE")
```

EXAMPLE 2.2...

```
      COMMON /STRINGS/ JS(50)
      CALL KINIT(9)
      ...
      CALL KONLST(JS(10),"+","-","*","/")
```

THIS EXAMPLE DEFINES ONLY JS(10) THROUGH JS(13) INCLUSIVE. THE REMAINING ELEMENTS OF THE ARRAY ARE NOT AFFECTED.

INTEGER FUNCTION KAT - CONCATENATE STRINGS

LEGAL FORMAT...

<STR.REP.>=KAT(<STR.REP.>,<STR.REP.>[,<STR.REP>...]) [3.1]

ARGUMENTS...

<STR.REP.> IS A STRING REPRESENTATION.

ALL OF THE STRINGS REPRESENTED BY THE ARGUMENTS ARE CONCATENATED IN THE SEQUENCE GIVEN, AND THE RESULTING REPRESENTATION RETURNED AS THE VALUE OF THE FUNCTION. ANY NUMBER OF ARGUMENTS (GREATER THAN 1) MAY BE USED, USING CONTINUATION CARDS IF NEEDED.

EXAMPLE 3.1...

```
      DIMENSION LS(4)
      CALL KINIT(7)
      ...
      CALL KONLST(LS," IS THE TIME"," AID OF"," PARTY"," NOW")
      JS=KON(" FOR ALL GOOD MEN TO COME TO ")
      NEWSTR=KAT(LS(4),LS(1),JS,KON(" THE"),LS(2),LS(3))
```

NEWSTR WILL THEN REPRESENT THE STRING,

" NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF PARTY"

INTEGER FUNCTION KSUB - SUBSTRING EXTRACTION

LEGAL FORMATS...

<STR.REP.> = KSUB(<STR.REP.>,<START.COL.>) [4.1]
<STR.REP.> = KSUB(<STR.REP.>,<START.COL.>,<NO.CHAR.>) [4.2]

ARGUMENTS...

<STR.REP.> IS A STRING REPRESENTATION.
<START.COL.> IS THE STARTING COLUMN FOR THE SUBSTRING EXTRACTION WITHIN THE STRING.
<NO.CHAR.> IS THE NUMBER OF CHARACTERS TO BE EXTRACTED.

IF FORM [4.1] IS USED, THE ENTIRE REMAINDER OF THE STRING FROM CHARACTER NUMBER <START. COL.> WILL BE EXTRACTED.
IF FORM [4.2] IS USED, THEN ONLY THE SPECIFIED NUMBER OF CHARACTERS WILL BE EXTRACTED.

EXAMPLE 4.1...

```
      CALL KINIT(0)
      ...
      JST=KON("MISD/SEAD, BLDG 353")
      IBLDG=KSUB(JST,7,2)
```

THE VARIABLE IBLDG WILL REPRESENT "BLDG 353".

EXAMPLE 4.2...

```
      DIMENSION IALPHA(26)
      CALL KINIT(0)
      ...
      ISTR=KON("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
      DO 10 I=1,26
   10 IALPHA(I)=KSUB(ISTR,I,1)
```

AFTER THE ABOVE CODE IS EXECUTED, IALPHA(1) WILL REPRESENT "A", IALPHA(10) WILL REPRESENT "J", ETC.

INTEGER FUNCTION LEN - LENGTH OF STRING

LEGAL FORMAT...

<INTEGER RESULT>=LEN(<STRING REP.>) [5.1]

ARGUMENTS...

<STRING REP.> IS A STRING REPRESENTATION.

FUNCTION LEN RETURNS THE LENGTH OF THE STRING REPRESENTED BY THE ARGUMENT.

EXAMPLE 5.1...

```
      CALL KINIT(9)
      ...
      KSTR=KON(" ALPHA")
      LSTR=KAT(KSTR,KON("BETA"))
      N=LEN(LSTR)
```

N WILL BE 10 AFTER EXECUTION OF THIS CODE.

LOGICAL FUNCTION LEXIC - COMPARE STRINGS

LEGAL FORMAT...

<.TRUE./.FALSE.>=LEXIC(<STRING REP.>,<KEY>,<STRING REP.>) [6.1]

ARGUMENTS...

<STRING REP.> IS A STRING REPRESENTATION.
<KEY> IS A LITERAL KEYWORD INDICATING THE OPERATOR TO BE USED FOR THE COMPARISON. <KEY> MAY ONLY BE ONE OF THE FOLLOWING:

"LT" "LE" "EQ" "NE" "GE" "GT"

THE KEYWORDS HAVE THE SAME MEANING AS THEIR REGULAR FORTRAN COUNTERPARTS.

FUNCTION LEXIC RETURNS THE LOGICAL VALUE .TRUE. OR .FALSE.. THE STRINGS REPRESENTED BY THE FIRST AND THIRD ARGUMENTS ARE COMPARED LEXICALLY. IF THE CONDITION INDICATED BY <KEY> IS MET, THEN LEXIC RETURNS .TRUE.. OTHERWISE, .FALSE. IS RETURNED. THE STRINGS NEED NOT BE OF THE SAME LENGTH TO BE COMPARED. THE STRINGS ARE COMPARED FOR DICTIONARY-ORDER USING THE DISPLAY CODE COLLATING SEQUENCE OF THE CDC COMPUTER. IN ORDER FOR TWO STRINGS TO BE CONSIDERED EQUAL, THEY MUST HAVE THE SAME LENGTH AND COMPOSITION. THUS "HI " AND "HI" ARE NOT EQUAL. IN FACT, "HI" WILL ALWAYS PRECEDED "HI " IN THE COLLATING PROCESS, SO "HI" IS LESS THAN "HI ". IT IS IMPORTANT TO REALIZE THAT YOU MAY **NOT** COMPARE TWO STRINGS **WITHOUT** USING FUNCTION LEXIC. TWO STRINGS WHICH ARE IDENTICAL IN EVERY RESPECT WILL NOT IN GERNERAL HAVE THE SAME STRING REPRESENTATION. AND IF YOU COMPARE THE REPRESENTATIONS WITH AN ORDINARY IF STATEMENT WITHOUT LEXIC, THEY WILL EVALUATE AS UNEQUAL. THE SOLE EXCEPTION IS THE NULL STRING, WHICH IS ALWAYS REPRESENTED BY THE NUMERIC VALUE 0.

NOTE... WHEN USING FUNCTION LEXIC, IT IS IMPORTANT TO REMEMBER TO DECLARE LEXIC AS TYPE **LOGICAL** IN THE CALLING PROGRAM.

EXAMPLE 6.1...

```
      LOGICAL LEXIC
      CALL KINIT(5)
      ...
      ISTR=KON("KOWALSKI")
      JSTR=KON("KOWAL")
      IF(LEXIC(JSTR,"LT",ISTR))GO TO 10
      ...
```

CONTROL WILL BE TRANSFERRED TO STATEMENT 10 AT THE IF STATEMENT.

EXAMPLE 6.

```
      PROGRAM SORT(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
C          PROGRAM TO READ 100 CARDS, SORT THEM ALPHABETICALLY,
C          AND PRINT THE SORTED LIST.
      LOGICAL LEXIC
      DIMENSION NSTR(100)
      CALL KINIT(5)
      DO 100 I=1,100
  100 NSTR(I)=KIN(5)
      DO 300 I=1,99
      IP1=I+1
      DO 200 J=IP1,100
      IF(LEXIC(NSTR(J),"GE",NSTR(I)))GO TO 200
      ITEMP=NSTR(I)
      NSTR(I)=NSTR(J)
      NSTR(J)=ITEMP
  200 CONTINUE
  300 CALL KOUT(NSTR(I),1)
      CALL KOUT(NSTR(100),1)
      STOP
      END
```

NOTE... THE ABOVE EXAMPLE USES A VERY INEFFICIENT SORTING TECHNIQUE FOR THE SAKE OF SIMPLICITY. SEE REFERENCE [1] FOR A DESCRIPTION OF BETTER METHODS.

INTEGER FUNCTION MATCH - FIND SUBSTRING MATCHING PATTERN

LEGAL FORMATS...

<COL.NO.>=MATCH(<S.R.>,<KEY>,<PAT.S.R.>) [7.1]
<COL.NO.>=MATCH(<S.R.>,<KEY>,<PAT.LIST S.R.>,<NO.PAT.>,<I MATCH>) [7.2]

ARGUMENTS...

<S.R> IS A STRING REPRESENTATION FOR THE STRING TO BE SEARCHED.
<KEY> IS A KEYWORD INDICATING THE TYPE OF SEARCH TO BE MADE.
SELECTED FROM THE FOLLOWING CHOICES.

"FIRST" "FIRSTNOT" "LAST" "LASTNOT"

<PAT.S.R> IS THE STRING REPRESENTATION OF THE PATTERN TO BE SEARCHED FOR.
<PAT.LIST S.R.> IS AN ARRAY OF REPRESENTATIONS OF PATTERNS TO BE SEARCHED FOR.
<NO.PAT> IS THE NUMBER OF ELEMENTS IN THE PATTERN ARRAY.
<I MATCH> IS RETURNED TO THE CALLING PROGRAM AS THE SUBSCRIPT OF <PAT.LIST S.R> WHICH RESULTED IN A SUCCESSFUL MATCH.

FORM [7.1] IS USED WHEN A SINGLE PATTERN IS TO BE SEARCHED FOR IN THE STRING. FORM [7.2] IS USED WHEN SEVERAL PATTERNS ARE TO BE TRIED IN SUCCESSION. THE KEYWORDS INDICATE THE DIRECTION OF THE SEARCH AND WHETHER THE NORMAL CONDITION OR INVERSE CONDITION CONSTITUTES A "SUCCESSFUL" MATCH.
"FIRST" SEARCHES LEFT TO RIGHT FOR THE FIRST OCCURRANCE OF ANY OF THE PATTERNS. "FIRSTNOT" SEARCHES FOR THE FIRST CHARACTER WHICH DOES NOT MATCH ANY OF THE PATTERNS.
"LAST" SEARCHES FOR THE LAST OCCURRANCE OF ANY OF THE PATTERNS IN THE STRING. "LASTNOT" SEARCHES FOR THE LAST CHARACTER WHICH DOES NOT MATCH ANY OF THE PATTERNS GIVEN.
THESE DISTINCTIONS ARE BEST CLARIFIED BY EXAMPLES.

EXAMPLE 7.1...

```
      ISTR=KON("THE VALUE OF X IS 3 IF Y IS 2 AND IF Z IS ,")
      ICOL=MATCH(ISTR,"FIRST",KON("IF"))
      LCOL=MATCH(ISTR,"LAST",KON("IF"))
```

THIS EXAMPLE RESULTS IN ICOL=21 AND LCOL=35.

EXAMPLE 7.2...

```
      DIMENSION IPAT(4)
      CALL KINIT(0)
      ...
      CALL KONLST(IPAT, "+", "-", "*", "/" )
      ISTR=KIN(5)
      ICOLOP=MATCH(ISTR,"FIRST",IPAT,4,IMATCH)
```

THIS EXAMPLE READS A CARD FROM TAPE5 AND SEARCHES FOR THE FIRST ARITHMETIC OPERATOR. ASSUMING THAT THE CARD CONTAINED,

"378.45*23.11+3.0"

THEN THE RESULTS WOULD BE THAT ICOLOP=7 AND IMATCH=3, BECAUSE THE FIRST COLUMN THAT MATCHED ANY OF THE LISTED PATTERNS WAS COLUMN 7, AND THE PATTERN IT MATCHED WAS IPAT(3).

EXAMPLE 7.4...

```
ILNB=MATCH(IST,"LASTNOT",KON(" "))
```

THIS EXAMPLE SEARCHES A STRING TO FIND THE LAST NON-BLANK CHARACTER IN THE STRING.

NOTES ON PATTERN MATCHING...

IF THE MATCH FAILS (THAT IS, THE SPECIFIED PATTERN MATCHING CONDITION IS NOT MET BY ANY OF THE PATTERNS), THE VALUE RETURNED BY FUNCTION MATCH IS 0. WHEN USING FORM [7.2], <I MATCH> IS MEANINGLESS WHEN "NOT" KEYWORDS ARE USED. SEARCHES USING "LASTNOT" ARE NOT RECOMMENDED FOR PATTERNS LONGER THAN ONE CHARACTER.

INTEGER FUNCTION KIN - INPUT A STRING

LEGAL FORMATS...

```
<STRING REP.> = KIN(<LFN>)                                    [8.1]
<STRING REP.> = KIN(<LFN>,<LENGTH>)                           [8.2]
<STRING REP.> = KIN(<LFN>,-<PADDING>)                         [8.3]
```

ARGUMENTS...

<LFN> IS THE LOGICAL FILE NAME. IT MAY BE EITHER A UNIT NUMBER OR LEFT-JUSTIFIED-WITH-ZERO-FILL FILE NAME.
<LENGTH> IS THE NUMBER OF CHARACTERS TO BE DEFINED FOR THE RESULT STRING.
<PADDING> IS THE NUMBER OF BLANKS TO BE APPENDED TO THE END OF THE STRING (DISTINGUSIHED FROM <LENGTH> BY THE MINUS SIGN).

IF FORM [8.1] IS USED THEN THE STRING RESULTING WILL BE THE LENGTH OF THE RECORD READ WITH ALL TRAILING BLANKS REMOVED. FOR EXAMPLE, IF THE INPUT DEVICE IS AN 80 COLUMN CARD READER AND THE LAST NON-BLANK COLUMN ON THE CARD READ IS COLUMN 60, THEN THE STRING REPRESENTED BY <STRING REP> WILL BE 60 CHARACTERS LONG. IF FORM [8.2] IS USED, THEN THE STRING WILL BE DEFINED AS <LENGTH> CHARACTERS LONG REGARDLESS OF THE NUMBER OF CHARACTERS IN THE ACTUAL RECORD. FOR EITHER FORM, THE MAXIMUM RECORD LENGTH IS LIMITED BY THE CDC OPERATING SYSTEM TO 150 CHARACTERS.

END-OF-FILE ENCOUNTERED USING EITHER FORM WILL CAUSE KIN TO RETURN THE NULL STRING. FOR FORMAT [8.1], AN ENTIRELY BLANK CARD WILL RETURN A STRING CONSISTING OF A SINGLE BLANK (" ").

EXAMPLE 8.1...

```
      PROGRAM TEXT(INPUT,TAPE5=INPUT,OUTPUT)
      CALL KINIT(0)
      ...
      IST =KIN(5)
      ...
```

THIS EXAMPLE READS A CARD FROM THE INPUT FILE. THE LENGTH OF THE STRING DEPENDS ON THE CONTENTS OF THE RECORD READ.

EXAMPLE 8.2...

```
      PROGRAM TEST(INPUT,OUTPUT)
      CALL KINIT(0)
      INP=5LINPUT
      ...
      ISTR=KIN(INP)
      IF(ISTR.EQ.0)STOP "END-OF-FILE ENCOUNTERED ON INPUT"
      ...
```

THIS EXAMPLE ILLUSTRATES HOW A LEFT-JUSTIFIED-WITH-ZERO-FILL FILE NAME MAY BE USED INSTEAD OF A UNIT NUMBER. IT ALSO ILLUSTRATES A METHOD FOR TESTING FOR END-OF-FILE. ALTERNATIVELY, THE USER COULD TEST FOR LEN(ISTR) EQUAL TO 0, WHICH ALSO WOULD BE .TRUE. WHEN END-OF-FILE WAS ENCOUNTERED.

EXAMPLE B.

```
      PROGRAM TEST3(INPUT,TAPE5=INPUT,OUTPUT,TAPE6=OUTPUT,TAPE1)
      CALL KINIT
      IFILE=6LTAPE1
      ...
      ISTR=KIN(IFILE,100)
      ...
```

REGARDLESS OF THE CONTENT OF THE RECORD READ, LEN(ISTR) WILL BE 100 CHARACTERS. IF THE RECORD READ WAS LONGER THAN 100 CHARACTERS, IT WILL BE TRUNCATED. IF IT WAS SHORTER THAN 100 CHARACTERS, IT WILL BE PADDED WITH BLANKS. THE ONLY EXCEPTION IS THAT THE NULL STRING WILL BE ASSIGNED TO ISTR IF END-OF-FILE IS ENCOUNTERED ON TAPE1.

SUBROUTINE KOUT - OUTPUT A STRING

LEGAL FORMATS...

```
CALL KOUT(<LFN>,<STRING REP.>)                                  [9.1]
CALL KOUT(<LFN>,<STRING REP.>,<CARRIAGE CON.>)                  [9.2]
```

ARGUMENTS...

<LFN> IS THE LOGICAL FILE NAME EXPRESSED EITHER AS AN INTEGER NUMBER, OR AS A LEFT-JUSTIFIED-WITH-ZERO-FILL FILE NAME.
<STRING REP.> IS THE REPRESENTATION OF THE STRING TO BE OUTPUT.
<CARRIAGE CON.> IS A CARRIAGE CONTROL INDICATOR, CHOSEN FROM THE LIST BELOW.

<CARRIAGE CON.>=...	MEANING...
1	SINGLE SPACE (NORMAL PRINTING)
2	DOUBLE SPACE (SKIP 1 LINE)
0	OVERPRINT PREVIOUS LINE
-1	SKIP TO TOP OF NEXT PAGE (FORM-FEED)
(OMITTED)	NO CARRIAGE CONTROL (1ST CHAR. USED FOR CARRIAGE CONTROL, IF APPLICABLE TO PHYSICAL DEVICE).

EXAMPLE 9....

```
      PROGRAM ONE(OUTPUT,TAPE6=OUTPUT)
      CALL KINIT(8)
      ...
      ISTR=KON("THIS IS A STRING.")
      CALL KOUT(6,ISTR,1)
      ...
```

THIS EXAMPLE WILL PRINT "THIS IS A STRING" ON THE NEXT LINE OF THE OUTPUT DEVICE.

INTEGER FUNCTION KINTEK - INPUT FROM TEKTRONIX TERMINAL

LEGAL FORMAT...

<STRING REF.> = KINTEK(<LENGTH>) (10.1)

ARGUMENTS...

<LENGTH> IS THE DESIRED LENGTH OF STRING TO BE INPUT. IF SPECIFIED AS 0, THEN THE RESULTING STRING LENGTH WILL BE THE LENGTH OF THE INPUT RECORD AFTER ALL TRAILING BLANKS ARE REMOVED. THE MAXIMUM ALLOWABLE <LENGTH> IS 133, LIMITED BY THE TEKTRONIX SOFTWARE (TCS PLOT-10).

THIS FUNCTION IS SIMILAR TO KIN, EXCEPT IT IS COMPATIBLE WITH THE TCS SOFTWARE AND MAY BE USED ANYWHERE TEKTRONIX USERS WOULD ORDINARILY CALL SUBROUTINE AINST. NOTE THAT <LENGTH> IS A REQUIRED PARAMETER.

EXAMPLE 10.1...

```
      PROGRAM T(OUTPUT,TAPE61,TAPE62)
      CALL INITT(300)
      CALL KINIT(0)
      ...
      CALL MOVABS(0,200)
      ISTR=KINTEK(0)
      ...
```

THIS EXAMPLE MOVES THE CURSOR TO COORDINATES (0,200) AND WAITS FOR THE TERMINAL USER TO ENTER A LINE FROM THE KEYBOARD. THE PROGRAM CONVERTS THE INPUT LINE TO A STRING REPRESENTATION, ISTR.

NOTE THAT THIS ROUTINE REQUIRES THE USE OF THE TEKTRONIX TCS SOFTWARE, SUCH AS THE TEK30 OR TEK48 LIBRARIES. TCS IS A REGISTERED TRADEMARK OF TEKTRONIX, INC.

SUBROUTINE KOUTEK - OUTPUT TO TEKTRONIX

LEGAL FORMATS...

CALL KOUTEK(<STRING REP.>) [11.1]
CALL KOUTEK(<STRING REP.>,<CARRIAGE CON.>) [11.2]

ARGUMENTS...

<STRING REP.> IS THE REPRESENTATION OF THE STRING TO BE OUTPUT.
<CARRIAGE CON.> IS CARRIAGE CONTROL CODE, CHOSEN FROM THE LIST BELOW.

<CARRIAGE CON.>	MEANING...
1	NEW LINE, THEN OUTPUT
2	NEW LINE, NEW LINE, OUTPUT
OMITTED	OUTPUT AT PRESENT CURSOR POSITION

THIS SUBROUTINE IS PROVIDED TO GIVE COMPATABILITY WITH TEKTRONIX TERMINAL USERS USING TCS (PLOT-10) SOFTWARE. IT IS FUNCTIONALLY EQUIVALENT TO STRING FUNCTION KOUT, BUT MAY BE USED ANYWHERE TCS SUBROUTINE AOUTST WOULD NORMALLY BE USED.

NOTE THAT THIS ROUTINE REQUIRES THE TCS SOFTWARE SUCH AS IS CURRENTLY AVAILABLE ON THE TEK36 OR TEK48 LIBRARY.

INTEGER FUNCTION INTVAL - STRING TO INTEGER CONVERSION

LEGAL FORMATS...

<INTEGER VAL.> = INTVAL(<STRING REP.>) [12.1]
<INTEGER VAL.> = INTVAL(<STRING REP.>,<IF ERROR>) [12.2]

ARGUMENTS...

<STRING REP.> IS A REPRESENTATION OF THE STRING TO BE CONVERTED.
<IF ERROR> IS AN INTEGER VARIABLE PREVIOUSLY ASSIGNED TO A
VALID STATEMENT NUMBER IN THE CALLING PROGRAM BY
A FORTRAN **ASSIGN** STATEMENT.

INTVAL PERFORMS STRING TO INTEGER CONVERSION. IN NORMAL
FORTRAN APPLICATION PROGRAMS, USERS ARE OFTEN UNAWARE THAT
SUCH CONVERSIONS TAKE PLACE, BECAUSE THE READ AND FORMAT STATEMENTS
PERFORM THE CONVERSION AUTOMATICALLY. FOR EXAMPLE,

```
      READ(5,100)IVAL
  100 FORMAT(I5)
```

CAUSES THE COMPUTER TO READ A CHARACTER STRING FROM UNIT 5.
AND CONVERT THE FIRST 5 CHARACTERS TO THE NUMERIC VALUE THEY
REPRESENT. TO ACCOMPLISH THE SAME OPERATION USING THE
CHARACTER STRING ROUTINES, WE MIGHT WRITE,

```
      IVAL=INTVAL(KIN(5))
```

THE PRIMARY DIFFERENCE IN THESE TWO EXAMPLES IS THAT THE NUMBER
COULD BE PUNCHED ANYWHERE ON THE CARD FOR THE SECOND CASE,
AND COULD BE WRITTEN WITH OR WITHOUT A DECIMAL POINT,
AS WE SHALL PRESENTLY SEE.

CONVERSION RULES...

1. THE STRING TO BE CONVERTED MAY CONTAIN ANY LEGAL REPRESENTATION
OF A NUMBER. IT MAY BE WRITTEN WITH OR WITHOUT A DECIMAL POINT,
SIGNED OR UNSIGNED, AND MAY EVEN HAVE AN EXPONENT OF 10 SPECIFIED
IN THE USUAL FORTRAN REPRESENTATION (E.G., "1.23E4").
2. THE STRING MAY HAVE ANY NUMBER OF LEADING BLANKS, WHICH ARE
IGNORED.
3. CONVERSION IS CONSIDERED FINISHED WHEN A TRAILING BLANK IS
ENCOUNTERED OR THE STRING IS EXHAUSTED, WHICHEVER COMES FIRST.
4. IF AN ILLEGAL CHARACTER IS ENCOUNTERED IN THE STRING,
CONVERSION IS STOPPED AND ONE OF THE FOLLOWING OCCURS...

A. IF <IF ERROR> WAS NOT SPECIFIED ON THE CALL, THEN
A FATAL DIAGNOSTIC ERROR MESSAGE IS ISSUED AND THE PROGRAM STOPS.

B. IF <IF ERROR> WAS SPECIFIED, THEN CONTROL IS PASSED
DIRECTLY TO THE DESIGNATED STATEMENT NUMBER IN THE CALLING
PROGRAM. THIS FEATURE ALLOWS THE USER COMPLETE CONTROL OVER

RECOVERY FROM CONVERSION ERRORS.
5. AN ENTIRELY BLANK STRING RETURNS 0 WITHOUT ERROR.

EXAMPLES OF STRINGS TO BE CONVERTED AND THE RESULT...

IF THE STRING IS...	THEN INTVAL RETURNS...
"23"	23
" -23 "	-23
" 0 ORANG S "	0
" 3ORANGES "	(ERROR)
" -200.0"	-200
" 854.63 "	854
" .2336E+3"	233
" +2 3"	2
" .75 "	0
" "	0
"23.251"	(ERROR)

EXAMPLE 12.1...

```
      PROGRAM TEST(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      CALL KINIT(5)
      ...
C         READ THE VALUE OF NRUN FROM ANYWHERE ON THE CARD...
      NRUN=INTVAL(KIN(5))
      IF(NRUN.EQ.0)STOP
      ...
```

THIS EXAMPLE SHOWS HOW AN INTEGER VALUE MAY BE READ FROM ANYWHERE ON A CARD. FIRST, KIN(5) INPUTS A STRING. THE STRING IS THEN CONVERTED INTO ITS NUMERIC VALUE BY INTVAL. SHOULD THERE BE INFORMATION ON THE CARD OTHER THAN A NUMBER, THE PROGRAM WILL STOP AND A FATAL ERROR MESSAGE ISSUED.

EXAMPLE 12.2...

```
      PROGRAM TEST2(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      CALL CONNEC(5)
      CALL CONNEC(6)
      CALL KINIT(5)
      ...
      ASSIGN 10 TO IERRED
      ...
  101 CONTINUE
C         READ USERS TYPED VALUE OF NRUN...
      NRUN=INTVAL(KIN(5),IERRED)
      IF(NRUN.EQ.0)STOP
      ...
   10 CONTINUE
C         COME HERE ONLY ON CONVERSION ERROR.
      CALL KOUT(6,KON("BAD DATA, TRY AGAIN."),1)
      GO TO 101
      ...
```

THIS PROGRAM SEGMENT IS SIMILAR TO EXAMPLE 12.1 EXCEPT
THAT THE PROGRAM IS INTERACTIVE, AND THEREFORE THE PROGRAM
SHOULD GIVE THE USER AN OPPORTUNITY TO MEND HIS WAYS IF
HE MAKES A TYPO ERROR WHILE ENTERING THE VALUE FOR NRUN AT
THE KEYBOARD. SHOULD INTVAL DETECT ILLEGAL INPUT, IT WILL NOT
RETURN VIA THE NORMAL PATH, BUT WILL INSTEAD BRANCH DIRECTLY
TO STATEMENT 10. THUS IF THE USER TYPED "12", INTVAL
WOULD RETURN NORMALLY, BUT IF HE TYPED, "1K", INTVAL WOULD
RETURN TO STATEMENT 10.

REAL FUNCTION RLVAL - STRING TO REAL CONVERSION

LEGAL FORMATS...

<REAL VAL.> = RLVAL(<STRING REP.>) [13.1]
<REAL VAL.> = RLVAL(<STRING REP.>,<IF ERROR>) [13.2]

ARGUMENTS...

<STRING REP> IS THE REPRESENTATION OF THE STRING TO BE CONVERTED.
<IF ERROR> IS AN INTEGER VARIABLE PREVIOUSLY ASSIGNED TO A
VALID STATEMENT NUMBER IN THE CALLING PROGRAM BY
A FORTRAN ASSIGN STATEMENT.

THE RULES OF CONVERSION GIVEN FOR INTVAL APPLY ALSO TO
RLVAL, WITH THE EXCEPTION THAT THE FRACTIONAL PART OF NUMBERS
IS RETAINED, SINCE RLVAL RETURNS A REAL VALUE. THE
STRING TO BE CONVERTED MAY STILL BE WRITTEN WITH OR WITHOUT A
DECIMAL POINT.

EXAMPLE 13.1...

```
      IST=KON("13")
      JST=KON(".361")
      X=RLVAL(ISTR)
      ...
      ASSIGN 20 TO IERR
      Y=RLVAL(KAT(ISTR,JSTR),IERR)
      ...
   20 CONTINUE
      PRINT 106
  106 FORMAT(* ILLEGAL VALUE FOR Y*)
      ...
```

IN THIS EXAMPLE, X WILL BE 13.0 AND Y WILL BE 13.361.

INTEGER FUNCTION KSR - NUMERIC TO STRING CONVERSION

LEGAL FORMATS...

<STRING REF.> = KSR(<VALUE>) [14.1]
<STRING REF.> = KSR(<VALUE>,<FTN.FMT.>) [14.2]

ARGUMENTS...

<VALUE> IS A REAL OR AN INTEGER VALUE TO BE CONVERTED.
<FTN.FMT.> IS A LITERAL STRING CONTAINING A LEGAL FORTRAN
FORMAT TO BE USED TO GOVERN THE CONVERSION.

KSR IS THE INVERSE FUNCTION OF FUNCTIONS INTVAL AND RLVAL.
IT ACCEPTS AS INPUT THE NUMERIC VALUE OF A REAL OR INTEGER
NUMBER AND RETURNS THE STRING REPRESENTATION OF THE NUMBER.
IN AN ORDINARY FORTRAN WRITE STATEMENT, THIS CONVERSION
IS PERFORMED AUTOMATICALLY BY THE FORMAT STATEMENT. FOR
EXAMPLE,

```
      WRITE(6,200) X
  200 FORMAT(F10.4)
```

THIS CODE CAUSES THE COMPUTER TO TAKE THE NUMERIC VALUE OF X
AND CONVERT IT TO A STRING WITH 1 CHARACTERS REPRESENTING
THE VALUE OF X, AND THEN OUTPUT IT TO THE DEVICE. TO
ACCOMPLISH THE SAME THING USING THE STRING ROUTINES, WE
COULD WRITE,

```
      CALL KOUT(6,KSR(X,"(F10.4)"))
```

THIS WILL CAUSE EXACTLY THE SAME RESULT AS THE WRITE STATEMENT.
THE MORE INTERESTING CASE THOUGH, IS THE CASE WHERE KSR IS
USED WITHOUT FORMAT SPECIFICATION, AS SHOWN IN FORM [14.1].
FOR THIS CASE, KSR WILL AUTOMATICALLY SELECT A FORMAT WHICH WILL
EXACTLY FIT THE MAGNITUDE OF THE NUMBER, REGARDLESS OF ITS VALUE.

CONVERSION RULES FOR KSR...

A. IF <FTN.FMT> IS NOT SPECIFIED...

1. IF <VALUE> IS AN INTEGER NUMBER, THEN KSR WILL SELECT AN
I-TYPE FORMAT WITH THE SMALLEST NUMBER OF CHARACTERS
POSSIBLE TO REPRESENT THE NUMBER.

2. IF <VALUE> IS A REAL NUMBER, KSR WILL SELECT A REAL
FORMAT REPRESENTATION AS FOLLOWS: IF <VALUE> IS BETWEEN
-1.E-5 AND +1.E9, AN F-TYPE SPECIFICATION WILL BE USED
WITH ALL LEADING BLANKS REMOVED AND ALL TRAILING
ZEROS SUPPRESSED FROM THE NUMBER. NUMBERS OUTSIDE THIS
RANGE WILL BE CONVERTED USING 1PE11.5 FOR POSITIVE VALUES
AND 1PE12.5 FOR NEGATIVE VALUES. THE ONLY EXCEPTION IS
THE EXACT VALUE 0.0, WHICH WILL BE CONVERTED TO "0", SINCE
THERE IS NO DISTINCTION BETWEEN A REAL 0.0 AND AN INTEGER 0
IN THE CDC REPRESENTATION OF VALUES.

B. IF <FT..FMT.> IS SPECIFIED...

1. ANY LEGAL FORMAT MAY BE USED WHICH DOES NOT CAUSE THE SYSTEM MAXIMUM RECORD LENGTH OF 15[illegible] CHARACTERS TO BE EXCEEDED.

2. ANY TRAILING BLANKS GENERATED BY THE FORMAT WILL BE IGNORED, FOR REASONS EXPLAINED IN THE DISCUSSION OF FUNCTION KON.

3. THE FORMAT NEED NOT AGREE IN TYPE WITH <VALUE>. FOR EXAMPLE IF A <VALUE> OF 13.5 IS SPECIFIED WITH A FORMAT OF "(I5)", KSR WILL AUTOMATICALLY CONVERT THE VALUE TO INTEGER BEFORE CONVERSION, SO THE STRING RESULT WOULD BE " 13".

EXAMPLE 14.1...

```
      PROGRAM TEST(OUTPUT,TAPE6=OUTPUT)
      CALL KINIT(0)
      ...
      N=1
      ...
      N=1000000
      ...
      WRITE(6,20)N
   20 FORMAT(* THERE WERE *,I7 * OCCURRANCES.*)
      ...
      ISTR=KAT(KON("THERE WERE",-1),KSR(N),KON(" OCCURRANCES."))
      CALL KOUT(6,ISTR,1)
      ...
```

THE ABOVE EXAMPLE ILLUSTRATES THE DIFFERENCES BETWEEN USING AN ORDINARY FORTRAN WRITE AND USING A FREE-FORMAT CONVERSION WITH FUNCTION KSR. ASSUMING N CAN RANGE IN VALUE FROM 0 TO 1000000 DEPENDING ON THE SITUATION, THE PROGRAMMER SPECIFIED AN I7 CONVERSION FOR N, WHICH IS THE MINIMUM SIZE WHICH WILL FIT ANY CONTINGENCY. NOW SUPPOSE N WAS 3. THE FORTRAN WRITE/FORMAT COMBINATION WOULD PRODUCE,

```
THERE WERE       3 OCCURRANCES.
```

THE KSR CONVERSION WOULD PRODUCE,

```
THERE WERE 3 OCCURRANCES.
```

WHICH IS MUCH MORE READABLE.

THE DYNAMIC FORMATTING CAPABILITY ILLUSTRATED ABOVE IS VERY USEFUL IN PREPARING ATTRACTIVE OUTPUT, ESPECIALLY IN INTERACTIVE APPLICATIONS. FLOATING-POINT NUMBERS MAY ALSO BE USED.

EXAMPLE 14.2...

```
      PROGRAM TEST(OUTPUT,TAPE6=OUTPUT)
      CALL KINIT( )
      ...
      XVAL=49./4.
      ...
      ISLASH=KON("///")
      ISTR=KAT(ISLASH,KSP(XVAL),ISLASH)
      CALL KOUT(6,ISTR,1)
      ...
```

THE PRINTED OUTPUT WOULD BE,

///12.25///

THIS EXAMPLE ILLUSTRATES THE ABILITY OF KSP TO SELECT SUITABLE FORMATS FOR REAL NUMBERS.

EXAMPLE 14.3...

```
      PROGRAM TEST(OUTPUT,TAPE6=OUTPUT)
      CALL KINIT( )
      ...
      X=75.0/2.
      ...
      CALL KOUT(6,KSP(X,"(F10.5)"))
```

THIS EXAMPLE ILLUSTRATES THE USE OF FORM (14.2) AND THE RESULTING PRINTOUT WOULD READ,

```
 37.50000
```

SUBROUTINE KREPL - REPLACE STRING WITH STRING

LEGAL FORMAT...

CALL KREPL(<STRING REP.>,<STRING REP.>) [15.1]

ARGUMENTS...

<STRING REP.> IS A STRING REPRESENTATION

KREPL REPLACES THE FIRST NAMED STRING WITH THE SECOND STRING. FOR EXAMPLE.

```
CALL KREPL(ISTR,KON(" NEW"))
```

CAUSES THE VARIABLE ISTR TO BE ASSIGNED THE REPRESENTATION OF " NEW". THIS SEEMS EQUIVALENT TO,

```
ISTR=KON(" NEW")
```

SO IT SEEMS REASONABLE TO QUESTION THE UTILITY OF SUBROUTINE KREPL. LATER WE SHALL SEE THAT KREPL IS MORE EFFICIENT IN MEMORY USAGE THAN USING THE "=" ASSIGNMENT FORM. **IT IS IMPORTANT TO USE KREPL INSTEAD OF SIMPLE ASSIGNMENT STATEMENT ANY TIME THE ASSIGNMENT IS IN A LOOP WHERE THE VARIABLE IS TO BE "RE-USED" OVER AND OVER.** OTHERWISE, THE SIMPLE ASSIGNMENT STATEMENT IS ACCEPTABLE.

NOTE THAT THE VARIABLE TO BE REPLACED MUST BE A LEGAL STRING REPRESENTATION. IT MAY, OF COURSE, BE THE NULL STRING.

EXAMPLE 15.1...

```
      PROGRAM COPY(INPUT,TAPE5=INPUT,OUTPUT,TAPE6=OUTPUT)
      CALL KINIT(5)
      INSTR=0
    1 CONTINUE
      CALL KREPL(INSTR,KIN(5))
      IF (INSTR.EQ.0)STOP "END-OF-INPUT "
      CALL KOUT(6,INSTR,1)
      GO TO 1
      END
```

THIS EXAMPLE ILLUSTRATES THE PROPER WAY TO CODE A PROGRAM TO READ AN INPUT DECK AND PRINT IT. IT IS IMPORTANT TO USE KREPL INSTEAD OF SIMPLY USING INSTR=KIN(5) BECAUSE THIS STATEMENT IS EXECUTED MANY TIMES. AS WE SHALL SEE LATER, THE AMOUNT OF STORAGE USED FOR STRINGS WILL BE THE LENGTH OF THE LONGEST LINE WHEN THE PROGRAM IS RUN IN THE FORM ABOVE. BUT WOULD BE THE SUM OF THE LENGTHS OF ALL THE LINES IF A SIMPLE REPLACEMENT STATEMENT WERE USED INSTEAD OF KREPL.

SUBROUTINE KINIT - INITIALIZE CHARACTER STRING ROUTINES

LEGAL FORMAT...

CALL KINIT(<BUF.SIZE>) [15.1]

ARGUMENTS...

<BUF.SIZE> IS THE DIMENSION OF THE MEMORY BUFFER TO BE USED FOR STORAGE OF ACTUAL-STRINGS. IF SPECIFIED AS 0, THE DEFAULT SIZE OF 512 WILL BE USED.

KINIT INITIALIZES THE COMMON VARIABLES SHARED AMONG THE VARIOUS FUNCTIONS AND SUBROUTINES IN THE CHARACTER STRING PACKAGE. NORMALLY, THE FORM OF THE CALL IS,

```
CALL KINIT()
```

OPTIONAL VALUES FOR <BUF.SIZE> WILL BE INTRODUCED IN THE NEXT SECTION.

DEFINITION OF <STRING REPRESENTATION>

UNTIL NOW, WE HAVE BEEN USING THE NOTION OF <STRING REP.> WITHOUT KNOWING WHAT IT IS. A <STRING REPRESENTATION> IS A SINGLE WORD (SIMPLE VARIABLE) WHICH UNIQUELY DESCRIBES A STRING OF CHARACTERS. THE READER WHO IS FAMILIAR WITH THE ARCHITECTURE OF THE CDC 6600 SERIES COMPUTER WILL KNOW THAT ONLY 10 CHARACTERS MAY BE STORED IN A SINGLE COMPUTER WORD. SINCE THE STRING ROUTINES ARE CAPABLE OF REPRESENTING STRINGS OF UP TO 500 CHARACTERS USING A SINGLE WORD, IT SHOULD BE APPARENT THAT IT IS IMPOSSIBLE TO STORE THE ACTUAL-STRING IN THE VARIABLE REPRESENTING THE STRING.

ACTUALLY THE <STRING REP.> IS JUST A POINTER TO A LOCATION IN MEMORY CONTAINING THE ACTUAL-STRING. THE ACTUAL-STRING IS A CONTIGUOUS BLOCK OF CHARACTERS OCCUPYING AS MANY WORDS AS NEEDED. ACTUAL STRINGS ARE NORMALLY ALLOCATED AUTOMATICALLY TO A RESERVED BLOCK OF MEMORY CALLED THE BUFFER.
WHEN A <STRING REP.> IS GENERATED AUTOMATICALLY BY A STRING FUNCTION OR SUBROUTINE, IT ALWAYS CONTAINS THE FOLLOWING SUBFIELDS WHICH ARE PACKED INTO THE SINGLE WORD <STRING REP.>:

BITS	SUBFIELD NAME	DESCRIPTION
0 - 8	<LENGTH>	NUMBER OF CHARACTERS IN ACTUAL-STRING.
9 - 26	<SUB>	SUBSCRIPT OF THE BUFFER TO ADDRESS 1ST CHARACTER OF ACTUAL-STRING.
27 - 47	<EXTENT>	THE DISK EXTENT (BLOCK) ON WHICH THE ACTUAL-STRING IS STORED.
48 - 53	<INTERLOCK>	ALWAYS SET TO 76B. A SOFTWARE INTERLOCK DESIGNATING THIS WORD AS A <STRING REP.>.
54 - 59	<CHAR.1>	DISPLAY CODE OF ACTUAL-STRING FOR STRINGS OF LENGTH 1. FOR ALL OTHER STRINGS, NOT USED AND SET TO 00B.

LET US SEE HOW THE STRING ROUTINES WOULD FORM A REPRESENTATION FOR INSTR, WHERE INSTR IS DEFINED BY,

```
INSTR=KIN(5)
```

FUNCTION KIN READS A STRING INTO MEMORY, REMOVES ALL TRAILING BLANKS, AND COPIES THE REMAINING STRING TO SOME LOCATION IN THE BUFFER. THIS BUFFER IS CALLED DSA, FOR "DYNAMIC STORAGE AREA". THE STRING REPRESENTATION ASSIGNED TO INSTR WILL CONSIST OF THE LENGTH OF THE STRING IN THE LOW ORDER 9 BITS AND THE SUBSCRIPT OF DSA NEEDED TO ADDRESS THE ACTUAL STRING IN THE NEXT 18 BITS. EACH TIME THIS STATEMENT IS EXECUTED, KIN WILL PUT THE NEW ACTUAL-STRING IN THE BUFFER AND UPDATE THE STRING REPRESENTATION FOR INSTR. IT WILL NOT, HOWEVER, DESTROY OR REPLACE THE OLD ACTUAL-STRING PREVIOUSLY ASSIGNED TO INSTR. THIS IS BECAUSE KIN IS A FUNCTION AND SO IT DOESNT "KNOW" WHERE ITS RETURNED VALUE WILL BE ASSIGNED. THERE IS NO WAY KIN CAN

DETERMINE THAT THE STRING REPRESENTATION WILL BE REPLACING INSTR. ON THE OTHER HAND, IF WE HAD CODED,

CALL KREPL(INSTR,KIN(-))

THEN KREPL WOULD KNOW THAT INSTR WAS TO BE REPLACED, AND SO IT WOULD RELEASE THE STORAGE SPACE USED BY THE OLD ACTUAL-STRING FOR INSTR. THIS EXPLAINS WHY IT IS IMPORTANT TO USE KREPL INSTEAD OF USING SIMPLE ASSIGNMENT STATEMENTS WHEN A VARIABLE IS RE-USED FOR <STRING REPRESENTATIONS>.

WHAT HAPPENS WHEN THE BUFFER AREA IS COMPLETELY FILLED? PREVIOUSLY THIS REPORT CLAIMED THAT ANY NUMBER OF STRINGS COULD BE STORED. NOW IT SEEMS THAT WE MIGHT RUN OUT OF SPACE IN THE BUFFER FOR ACTUAL-STRINGS. IN FACT, IF THE BUFFER IS SMALL, WE SHALL RUN OUT OF SPACE RATHER QUICKLY. WHEN THIS HAPPENS, THE CHARACTER STRING ROUTINES AUTOMATICALLY COPY THE CURRENT BUFFER ONTO DISK AND CLEAR THE ENTIRE DSA FOR NEW ACTUAL-STRINGS. EACH TIME THE BUFFER IS COMPLETELY FILLED, ANOTHER BLOCK OF DISK SPACE IS AUTOMATICALLY ALLOCATED FOR ITS STORAGE. EACH OF THESE BLOCKS IS CALLED AN <EXTENT>, AND IS IDENTIFIED BY A UNIQUE NUMBER. THE EXTENT BECOMES AN INTEGRAL PART OF THE <STRING REP.>, SO THAT WHEN AN ACTUAL-STRING IS NEEDED BY ONE OF THE STRING ROUTINES, IT LOOKS TO SEE IF THE STRING WAS DEFINED IN THE SAME <EXTENT> AS THE <EXTENT> WHICH IS CURRENTLY IN THE BUFFER. IF NOT, THE CORRECT <EXTENT> IS RECALLED FROM DISK INTO THE DSA. THIS PROCEDURE IS DONE AUTOMATICALLY WITH A RANDOM ACCESS DISK FILE CALLED KFILE. KFILE IS GENERATED WHEN KINIT IS FIRST CALLED, AND SHOULD **NOT** BE DECLARED ON THE PROGRAM CARD. ALTHOUGH THE DISK OPERATIONS PERFORMED ARE MUCH MORE EFFICIENT THAN WOULD BE POSSIBLE USING ORDINARY FORTRAN READ AND WRITE STATEMENTS, A DISK ACCESS STILL TAKES ABOUT **1000** TIMES LONGER THAN A MEMORY ACCESS. THEREFORE IT SHOULD BE APPARENT THAT A BIG BUFFER (DSA) IS DESIRABLE IF A LOT OF CHARACTER STRING OPERATIONS ARE TO BE PERFORMED. SINCE THE LARGER ARRAY DSA IS, THE FEWER DISK ACCESSES WILL BE REQUIRED. OF COURSE, IF MEMORY IS AT A PREMIUM IN A CERTAIN APPLICATION, THE ROUTINES WILL STILL WORK WITH A DSA AS SMALL AS 64 WORDS. REGARDLESS OF THE SIZE OF THE DSA BUFFER, THE TOTAL STORAGE CAPACITY FOR ACTUAL-STRINGS WILL BE SEVERAL MILLION CHARACTERS. IN ALL CASES THE USER WILL RUN OUT OF MEMORY FOR STORAGE OF <STRING REPRESENTATIONS> BEFORE STORAGE FOR ACTUAL-STRINGS IS EXHAUSTED, SUPPORTING THE CLAIM TO VIRTUALLY UNLIMITED STORAGE.

IN ORDER TO CONSERVE SPACE IN THE BUFFER, SEVERAL SPECIAL CASES ARE DEFINED FOR FORMING STRING REPRESENTATIONS:

1. IF THE ACTUAL-STRING IS ONLY ONE CHARACTER LONG, THEN THE ACTUAL STRING WILL NOT BE ALLOCATED IN THE BUFFER, BUT WILL BE STORED DIRECTLY IN THE UPPER 6 BITS OF THE <STRING REP.>.

SINGLE CHARACTER STRINGS ARE FREQUENTLY ENCOUNTERED AND THIS SAVES BOTH MEMORY SPACE AND ACCESS TIME.

2. STRING CONSTANTS (DEFINED BY FUNCTION KON AND SUBROUTINE KONLST) ARE NOT COPIED INTO THE DSA BUFFER BECAUSE THE CONTENTS OF A STRING CONSTANT SHOULD NEVER CHANGE. THE ACTUAL-STRING USED IS THE ACTUAL-STRING PASSED AS AN ARGUMENT, AND THE VALUE OF <SUB> IS THE SUBSCRIPT OF DSA NECESSARY TO ADDRESS THE ACTUAL-STRING IN THE USER'S PROGRAM. IT SHOULD BE NOTED THAT <SUB> MAY BE A NEGATIVE NUMBER IN THIS CASE IF THE LOCATION OF THE ACTUAL-STRING IS A MEMORY ADDRESS LOWER THAN THE ADDRESS OF DSA(1). A CONSEQUENCE OF THIS SYSTEM IS THAT **PROGRAMS USING OVERLAYS MUST MAINTAIN THE OVERLAY DEFINING A GIVEN STRING CONSTANT IN CORE WHENEVER THAT STRING CONSTANT IS REFERENCED BY ITS <STRING REP.>.**

THE SIX BITS COMPRISING <INTERLOCK> ARE A PATTERN OF BITS WHICH WOULD NOT NORMALLY BE FOUND IN THAT POSITION IN ANY REAL OR INTEGER NUMBER. ALL STRING ROUTINES WHICH DEFINE A STRING RESULT SET THIS PATTERN, AND ALL ROUTINES EXPECTING A <STRING REP.> FOR AN ARGUMENT CHECK FOR IT. IF THE INTERLOCK PATTERN DOES NOT EXIST, THE DIAGNOSTIC "ILLEGAL STRING REPRESENTATION IS ISSUED. THIS HELPS TRAP BOGUS STRING REPRESENTATIONS WHICH WOULD OTHERWISE YIELD UNPREDICTABLE AND USUALLY DISASTEROUS ERRORS.

SUBROUTINE KINIT REVISITED

NOW THAT WE UNDERSTAND HOW ACTUAL-STRINGS ARE ALLOCATED TO MEMORY AND DISK, WE ARE READY TO REVISIT SUBROUTINE KINIT TO SEE HOW WE MAY ALTER THE SIZE OF THE BUFFER IF WE WISH. SUBROUTINE KINIT HAS A DEFAULT BUFFER IN LABELLED COMMON AS FOLLOWS...

```
COMMON / KSPACE / DSA(512)
```

IN ORDER TO CHANGE THE BUFFER SIZE, ALL WE HAVE TO DO IS RE-DEFINE THE COMMON BLOCK IN THE MAIN PROGRAM, AND USE THE NEW DIMENSION IN THE CALL TO KINIT. FOR EXAMPLE, TO INCREASE THE BUFFER SIZE FROM 512 TO 2048 WORDS,

```
PROGRAM BIG(INPUT,OUTPUT)
COMMON / KSPACE / DSA(2048)
CALL KINIT(2048)
...
```

THE DIMENSION CHOSEN FOR DSA SHOULD BE AN EXACT MULTIPLE OF 64. WHEN THIS METHOD IS USED, A NUMBER OF NON-FATAL LOADER ERRORS ARE GENERATED:

```
COMMON BLOCK REDEFINITION - KSPACE
LAST PROGRAM READ - XXXXX
LAST FILE ACCESSED - XXXXX
```

THESE DIAGNOSTICS MAY BE IGNORED.

SUBROUTINE KRLS - RELEASE STRING SPACE NO LONGER NEEDED

LEGAL FORMAT...

CALL KRLS(<STRING REP.>.[,<STRING REP.>...]) [17.1]

ARGUMENTS...

<STRING REP.> IS THE STRING REPRESENTATION OF THE STRING WHICH
IS NO LONGER NEEDED.

THIS ROUTINE MAY BE CALLED WITH ANY NUMBER OF ARGUMENTS.
THE STRING SPACE IN THE BUFFER ALLOCATED FOR THE STRINGS WHOSE
REPRESENTATIONS ARE PASSED AS ARGUMENTS WILL BE FREED-UP FOR RE-USE.

EXAMPLE 17.1...

```
CALL KRLS(ISTR.JSTR,KSTR)
```

THIS CALL WILL FREE SPACE OCCUPIED BY ISTR.JSTR, AND KSTR.
IF THE ACTUAL-STRINGS FOR ANY OF THESE STRINGS ARE STORED
IN A LOCATION OTHER THAN ARRAY DSA, THAT PARAMETER WILL BE IGNORED.

SUBROUTINE KRLSL - RELEASE SPACE FOR ARRAY OF STRINGS

LEGAL FORMAT...

CALL KRLSL(<STR.REP.ARRAY>,<NO.RLS.>) [18.1]

ARGUMENTS...

<STR.REP.ARRAY> IS THE FIRST ELEMENT OF THE ARRAY TO BE RELEASED.
<NO.RLS.> IS THE NUMBER OF ELEMENTS TO BE RELEASED.

KRLS MAY BE USED TO RELEASE THE SPACE USED FOR STORAGE
OF ACTUAL STRINGS FOR ALL OR PART OF AN ARRAY OF STRINGS.

EXAMPLE 18.1...

```
      PROGRAM TEST(INPUT,OUTPUT)
      DIMENSION ISTR(100)
      CALL KINIT(0)
      ...
      DO 100 I=1,100
  100 ISTR(I)=KIN(5)
      ...
C         LAST 40 STRINGS NO LONGER NEEDED. FREE-UP SPACE.
      CALL KRLSL(ISTR(61),40)
      ...
```

THE CALL TO KRLSL WILL FREE BUFFER SPACE IN ARRAY DSA PREVIOUSLY
OCCUPIED BY THE ACTUAL-STRINGS FOR ISTR(61) THROUGH ISTR(100).
THE SPACE RECLAIMED WILL BE USED BY THE STRING ROUTINES FOR
STORAGE OF NEW ACTUAL-STRINGS AS NEEDED.

INTEGER FUNCTION KDC - DECODE ACTUAL-STRING TO STRING REPRESENTATION

LEGAL FORMAT...

<STRING REP.> = KDC(<ACTUAL-STRING>,<COL.1>,<LENGTH>) [19.1]

ARGUMENTS...

<ACTUAL-STRING> IS A VARIABLE, ARRAY, OR LITERAL STRING CONTAINING
THE ACTUAL STRING TO BE DECODED.
<COL.1> IS THE NUMBER OF THE FIRST CHARACTER IN THE <ACTUAL-STRING>
TO BE DECODED.
<LENGTH> IS THE NUMBER OF CHARACTERS TO BE DECODED.

KDC CAN BE USED IN A MANNER SIMILAR TO THE FORTRAN EXTENDED
DECODE STATEMENT. KDC WILL RETURN THE STRING REPRESENTATION
OF A STRING OF DISPLAY-CODE CHARACTERS PACKED 10 TO THE WORD,
STARTING IN CHARACTER POSITION <COL.1>, <LENGTH> CHARACTERS
LONG. THIS SUBSTRING IS COPIED INTO THE CHARACTER ROUTINES BUFFER
PRIOR TO GENERATION OF THE STRING REPRESENTATION.

EXAMPLE 19.1

```
      PROGRAM TEST(INPUT,TAPE5=INPUT,OUTPUT,TAPE6=OUTPUT)
      DIMENSION IBCD(1 )
      CALL KINIT( )
      ...
      READ(5,20)(IBCD(L),L=1,8)
   20 FORMAT(7A10,A2)
      ...
      ISTR=KDC(IBCD,21,20)
     ...
```

THIS EXAMPLE READS A CARD IMAGE INTO ARRAY IBCD IN DISPLAY CODE
FORMAT, AND THEN EXTRACTS THE CONTENTS OF COLUMN 21 THROUGH 40
AS STRING REPRESENTATION ISTR.

SUBROUTINE KEC - ENCODE STRING REPRESENTATION TO ACTUAL-STRING

LEGAL FORMAT...

CALL KEC(<STRING REP.>,<DEST.>,<COL.1>) [20.1]

ARGUMENT LIST...

<STRING REP.> IS THE REPRESENTATION FOR THE STRING TO BE ENCODED.
<DEST.> IS A VARIABLE OR ARRAY TO RECEIVE THE ACTUAL STRING.
<COL.1> IS THE CHARACTER POSITION IN THE <DEST.> WHERE THE
ACTUAL-STRING IS TO START.

THIS SUBROUTINE PROVIDES THE INVERSE FUNCTION TO FUNCTION KDC,
AND IS SIMILAR TO THE FORTRAN EXTENDED ENCODE STATEMENT.
THE DISPLAY CODE ACTUAL-STRING REPRESENTED BY <STRING REP.> IS
COPIED INTO THE DESTINATION ARRAY OR VARIABLE STARTING AT <COL.1>.
THE NUMBER OF CHARACTERS ENCODED WILL BE LEN(<STRING REP.>).
CHARACTERS IN <DEST.> BEFORE <CHAR.1> AND AFTER THE END OF THE
ACTUAL-STRING WILL NOT BE AFFECTED.

EXAMPLE 20.1...

```
      PROGRAM TEST(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      DIMENSION IBCD(1 )
      CALL KINIT( )
      ...
      N=27
      ENCODE(40,105,IBCD)N
  105 FORMAT(* OF THE *,I2,* ENTRIES *)
      ...
      J=11
      ISTR=KAT(KSR(J),KON(" WERE VALID."))
      CALL KEC(ISTR,IBCD,20)
      PRINT 200,(IBCD(L),L=1,4)
  200 FORMAT(3A10,A5)
      ...
```

THIS EXAMPLE WILL RESULT IN THE PRINTING OF,

OF THE 27 ENTRIES 11 WERE VALID.

APPENDIX A

CONTROL CARDS TO GAIN ACCESS TO KSPLIB ROUTINES

THE FOLLOWING SEQUENCE OF CONTROL CARDS SHOULD BE EXECUTED FOR BATCH PROGRAMS PRIOR TO LOADING THE FORTRAN OBJECT PROGRAM:

```
ATTACH,KSPLIB,ID=CARBREY.
LIBRARY,KSPLIB.
```

EXAMPLE PROGRAM DECK FOR BATCH JOB...

```
MYJOB.
COMMENT.(XXX-YYY,0000000),MYNAME  BILLING INFO CARD
FTN(ER,A,T)
ATTACH,KSPLIB,ID=CARBREY.
LIBRARY,KSPLIB.
LGO.
7/8/9
        PROGRAM CHARA(INPUT,OUTPUT,TAPE6=OUTPUT)
        CALL KINIT(1)
        ...
        END
7/8/9
        ...
        DATA CARDS USED BY PROGRAM CHARA
        ...
6/7/8/9
```

INTERACTIVE PROGRAMS COMPILED USING INTERCOM MAY USE THE FOLLOWING SEQUENCE OF COMMANDS:

```
ATTACH,KSPLIB,ID=CARBREY.
RUN,F,N.
XEQ,LDSET=LIB=KSPLIB,LOAD=LGO.
```

THE SOURCE DECK FOR THE ROUTINES DESCRIBED IN THIS MANUAL HAS NOT BEEN INCLUDED DUE TO ITS LENGTH. THE ROUTINES ARE WRITTEN IN CDC FORTRAN EXTENDED AND IN CDC COMPASS ASSEMBLY LANGUAGE.

REQUESTS FOR COPIES OF THE SOURCE DECK SHOULD BE MADE IN WRITING TO CHIEF, SCIENTIFIC AND ENGINEERING APPICATIONS DIVISION, MANAGEMENT INFORMATION SYSTEMS DIRECTORATE, ARRADCOM, DOVER, N.J.. AFTER DISTRIBUTION IS APPROVED, THE ROUTINES WILL BE PROVIDED ON A USER-SUPPLIED 7 OR 9 TRACK MAGNETIC TAPE.

APPENDIX B

SAMPLE PROGRAMS USING KSPLIB ROUTINES

THE FOLLOWING PAGES GIVE TWO SAMPLE PROGRAMS. A BRIEF EXPLANATION OF EACH PROGRAM IS PROVIDED, ALONG WITH A LISTING OF THE PROGRAM AND THE OUTPUT FROM A SAMPLE RUN.

EXAMPLE PROGRAM 1 - FOUR-FUNCTION CALCULATOR

THIS PROGRAM INTERPRETS ARITHMETIC EXPRESSIONS IN A FORM AS WOULD BE USED ON AN ORDINARY FOUR-FUNCTION CALCULATOR. FOR EXAMPLE, IF THE USER TYPES,

```
3*1.20 +17 =
```

THEN THE PROGRAM WILL PRINT,

```
20.60
```

THIS PROGRAM ANALYZES THE EXPRESSION LEFT TO RIGHT, WITHOUT ANY HIERARCHY OF OPERATORS, UNTIL AN "=" SIGN IS ENCOUNTERED. THE RESULT IS THEN PRINTED. PARENTHESES ARE NOT PERMITTED. THE PROGRAM MAY BE RUN EITHER IN BATCH MODE OR INTERACTIVELY PROVIDED INPUT AND OUTPUT FILES ARE CONNECTED. LIKE ALL THE EXAMPLE PROGRAMS IN THIS MANUAL, LITTLE OR NO ERROR-CHECKING IS PERFORMED IN THE INTEREST OF SIMPLICITY.

THE OPERATION OF THE PROGRAM IS SIMPLE. THE FIVE OPERATORS ARE DEFINED BY A KONLST CALL. THE PROGRAM SEARCHES THE INPUT LINE CHARACTER-BY-CHARACTER FOR AN OPERATOR. WHEN IT IS FOUND, IOPN IS SET EQUAL TO THE SUBSCRIPT OF THE OPERATOR ARRAY CORRESPONDING TO THE NEW OPERATOR. ALL THE COLUMNS UP TO BUT NOT INCLUDING THE OPERATOR ARE ASSUMED TO CONTAIN THE OPERAND, WHICH IS CONVERTED FROM STRING TO NUMERIC BY FUNCTION RLVAL. THE PORTION OF THE STRING PROCESSED IS DISCARDED BY EXTRACTION OF THE REMAINDER OF THE STRING, AND THE PROCESS REPEATED. EACH TIME BOTH OPERANDS AND THE OPERATOR HAVE BEEN DETERMINED, THE RESULT IS COMPUTED. WHEN "=" IS REACHED, THE RUNNING TOTAL IS PRINTED USING THE DEFAULT CONVERSION OF FUNCTION KSR.

ALTHOUGH THIS ENTIRE PROGRAM REQUIRES LESS THAN 30 STATEMENTS TO IMPLEMENT USING THE CHARACTER STRING PACKAGE, IT WOULD TAKE MORE THAN A THOUSAND LINES OF CODE TO IMPLEMENT THE EQUIVALENT PROGRAM WITHOUT THE STRING ROUTINES.

***** FORTRAN LISTING OF EXAMPLE PROGRAM 1 *****

```
      PROGRAM FUNK4(INPUT,TAPE5=INPUT,OUTPUT,TAPE6=OUTPUT)
C
C           ***** FOUR-FUNCTION CALCULATOR EMULATION PROGRAM *****
C
C        THIS PROGRAM SOLVES ARITHMETIC EXPRESSIONS ENTERED AS YOU
C        WOULD ENTER THEM ON AN ORDINARY 4-FUNCTION CALCULATOR. E.G.,
C        "2*3.74 - 2.6+ 18=", AND PRINTS THE VALUE OF THE RESULT, E.G.,
C        "22.88".  EVALUATION PROCEEDS LEFT-TO-RIGHT WITHOUT HIERARCHY
C        OF OPERATORS, UNTIL "=" IS ENCOUNTERED.  PARENTHESES ARE NOT
C        PERMITTED.  PROGRAM HALTS WHEN "STOP" ENCOUNTERED.
C
      LOGICAL LEXIC
      DIMENSION IOPER(5)
C        INITIALIZE PROGRAM.  SET STRINGS TO NULL.  DEFINE OPERATORS.
      CALL KINIT(0)
      ISTR=
      JSTR=
      CALL KONLST(IOPER,"+","-","*","/","=")
      ISTOP=KON("STOP")
    1 CONTINUE
C        READ NEXT EXPRESSION, HALT IF "STOP" ENTERED.
      ANS=0.
      IOP=0
      CALL KREPL(ISTR,KIN(5))
      IF(LEXIC(ISTR,"EQ",ISTOP))STOP
      CALL KREPL(JSTR,ISTR)
   11 CONTINUE
C        LOCATE NEXT OPERATOR IN STRING.  IOPN IS SUBSCRIPT OF IOPER.
      ICOLOP=MATCH(ISTR,"FIRST",IOPER,5,IOPN)
      IF(IOPN.EQ.0)STOP"ILLEGAL OR NULL EXPRESSION."
C        EVALUATE NUMBER PRECEEDING OPERATOR.
      VAL = RLVAL(KSUB(ISTR,1,ICOLOP-1))
C        UPDATE ANSWER.
      IF(IOP.EQ.0)ANS=VAL
      IF(IOP.EQ.1)ANS=ANS+VAL
      IF(IOP.EQ.2)ANS=ANS-VAL
      IF(IOP.EQ.3)ANS=ANS*VAL
      IF(IOP.EQ.4)ANS=ANS/VAL
C        DISCARD PORTION OF EXPRESSION ALREADY EVALUATED.
      CALL KREPL(ISTR,KSUB(ISTR,ICOLOP+1))
      IOP=IOPN
      IF(IOP.NE.5)GO TO 11
C        COME HERE WHEN "=" ENCOUNTERED.  PRINT RESULT.
      CALL KREPL(JSTR,KAT(JSTR,KSR(ANS)))
      CALL KOUT(6,JSTR,1)
      GO TO 1
      END
```

SAMPLE OUTPUT FROM PROGRAM FUNK4 RUN

```
2 + 2 =4.
    2*3 +1.28318=7.28318
100/3 + 50 -.6666 =82.6667333333
26 + 23 + 21+20+17+101-117=91.
2*222.1 - 100.2  =344.
```

SAMPLE PROGRAM 2 - ALGEBRAIC ASSIGNMENT STATEMENT INTERPRETER

THIS PROGRAM IS A MUCH-EXPANDED VERSION OF PROGRAM 1. THIS PROGRAM INTERPRETS COMPLETE ALGEBRAIC ASSIGNMENT STATEMENTS SIMILAR TO FORTRAN ASSIGNMENT STATEMENTS, INCLUDING PARENTHESES AND VARIABLES. FOR EXAMPLE, IF THE INPUT LINE READ,

```
XDISPERSION = 1 / (2 + 1.577)
```

THEN THE PROGRAM WOULD PRINT,

```
XDISPERSION = .27956388E-347
```

IF THE NEXT INPUT LINE READ,

```
YDISPERSION =1 + 2*((1-XDISPERSION)/(XDISPERSION**.707))
```

THEN THE RESULT PRINTED WOULD READ,

```
YDISPERSION = 4.54783459936
```

THE PROGRAM WILL AUTOMATICALLY REMEMBER THE VALUE OF THE VARIABLE XDISPERSION FROM THE PREVIOUS EQUATION. THE OPERATORS ALLOWED INCLUDE +, - (INCLUDING UNARY NEGATION), *, /, **, AND PARENTHESES. THE HIERARCHY OF EVALUATION IS THE SAME AS FORTRAN (E.G., MULTIPLIES DONE BEFORE ADDS IN THE ABSENCE OF PARENTHESES). VARIABLES MAY BE ANY LENGTH. UNLIKE FORTRAN, HOWEVER, IMBEDDED BLANKS ARE NOT PERMITTED IN NUMERIC CONSTANTS OR VARIABLES. NO DISTINCTION IS MADE BETWEEN REAL AND INTEGER VARIABLES OR CONSTANTS: ALL ARITHMETIC IS PERFORMED IN FLOATING POINT MODE. AS IN THE FIRST EXAMPLE, ALMOST NO ERROR CHECKING IS DONE IN ORDER TO KEEP THE PROGRAM REASONABLY SIMPLE.

IN ADDITION TO ILLUSTRATING SOME TECHNIQUES FOR USAGE OF THE STRING ROUTINES, THIS PROGRAM CAN ALSO SERVE AS A PEDAGOGICAL TOOL FOR TECHNIQUES USED IN HIGH-LEVEL LANGUAGE COMPILERS. THE METHOD USED FOR INTERPRETING THE ASSIGNMENT STATEMENTS IN THIS PROGRAM IS FUNDAMENTALLY THE SAME AS THE TECHNIQUE USED IN MOST MODERN HIGH-LEVEL LANGUAGE COMPILERS. REFERENCE [2] GIVES FURTHER INFORMATION.

THERE ARE TWO BASIC TOOLS USED IN THIS PROGRAM THAT MAKE POSSIBLE THE TRANSLATION PROCESS: POLISH NOTATION AND THE PUSHDOWN STACK. BECAUSE OF THE PROLIFERATION OF HEWLITT-PACKARD CALCULATORS, IT IS ASSUMED THAT THE PRINCIPLES OF OF POLISH NOTATION ARE WELL-KNOWN TO THE READER. TO REVIEW BRIEFLY, POLISH NOTATION IS A SYSTEM ALLOWING US TO WRITE AN EXPRESSION OF ARBITRARY COMPLEXITY WITHOUT PARENTHESES AND WITHOUT AMBIGUITY. FOR EXAMPLE, AN ALGEBRAIC (OR "INFIX") EXPRESSION SUCH AS,

```
X=2*(3+4)
```

COULD BE WRITTEN IN POLISH NOTATION AS,

3 4 + 2 * x =

THIS FORM IS USUALLY CALLED REVERSE POLISH NOTATION (RPN) SINCE THE OPERATORS FOLLOW THE OPERANDS. THE ADVANTAGE OF POLISH NOTATION IS THAT THE EXPRESSION IS SIMPLY SCANNED FROM LEFT TO RIGHT AND EVALUATED AS EACH OPERATOR IS ENCOUNTERED. USERS OF POCKET CALCULATORS BASED ON RPN ARE ALREADY WELL AQUAINTED WITH THE PRINCIPLES OF POLISH NOTATION, ALTHOUGH THEY MAY NOT BE AWARE OF IT. THESE USERS MAY ALSO BE AWARE OF THE TOOL THAT MAKES THE EXECUTION OF THE POLISH EXPRESSION POSSIBLE: THE PUSH DOWN STACK. THE OPERATION OF A PUSHDOWN STACK IS ILLUSTRATED IN FIGURE B-1. THE STACK MAY BE VISUALIZED AS A STACK OF TRAYS IN A CAFETERIA. THE LAST TRAY PLACED ON THE STACK IS ALWAYS THE FIRST ONE TAKEN OFF. SUCH A STACK IS SOMETIMES CALLED A LAST-IN-FIRST-OUT STACK, OR LIFO. TO EXECUTE A POLISH EXPRESSION, PROCEED AS FOLLOWS:

1. MOVING LEFT TO RIGHT, IF AN OPERAND IS ENCOUNTERED, PUSH IT ON THE STACK.
2. WHEN AN OPERATOR IS ENCOUNTERED, POP THE LAST TWO OPERANDS OFF THE STACK AND PERFORM THE INDICATED OPERATION ON THEM.
3. PUSH THE RESULT BACK ON THE STACK, FORMING A NEW OPERAND.

WHEN USING AN RPN CALCULATOR, THE "ENTER" KEY IS USED TO PUSH AN OPERAND ON THE STACK. THE OTHER OPERATIONS ARE PERFORMED AUTOMATICALLY BY THE CALCULATOR.

THE PUSH-DOWN STACK, WHICH IS THE KEY TO EXECUTING POLISH NOTATION, IS ALSO THE KEY TO CONVERTING INFIX NOTATION TO POLISH. FIGURE B-3 ILLUSTRATES A FLOWCHART FOR CONVERTING ALGEBRAIC EXPRESSIONS TO POLISH EXPRESSIONS USING A STACK. FIGURE B-4 SHOWS HOW WE MAY USE THE SAME STACK TO EXECUTE THE POLISH EXPRESSION. THUS WE HAVE A TWO-STEP CONVERSION PROCESS WHICH CAN HANDLE EXPRESSIONS OF ARBITRARY COMPLEXITY. FIGURE B-1 ALSO ILLUSTRATES HOW A PUSH-DOWN STACK IS IMPLEMENTED IN SOFTWARE. THE STACK IS JUST AN ARRAY WITH A POINTER MAINTAINED ELSEWHERE WHICH ALWAYS POINTS TO THE TOP OF THE STACK. WHEN THE STACK IS PUSHED, THE POINTER IS INCREMENTED. WHEN THE STACK IS POPPED, THE ARRAY ELEMENT POINTED TO IS RETURNED AND THE POINTER DECREMENTED. IN THE PROGRAM LISTING, THREE STACK ROUTINES ARE PROVIDED: IPUSH, WHICH PUSHES THE ARGUMENT ON THE STACK; IPOP, WHICH RETURNS THE ITEM POPPED OFF THE STACK, AND ISEE, WHICH ALLOWS US TO EXAMINE THE CONTENT OF THE TOP OF THE STACK WITHOUT POPPING IT. THE FLOW CHART IN FIGURE B-2 ILLUSTRATES THE OVERALL PROGRAM FLOW.

THE VARIABLES USED IN THE SAMPLE PROGRAM ARE STORED IN AN ARRAY AS ENCOUNTERED. IN PRACTICE, THIS METHOD IS TOO INEFFICIENT FOR REAL COMPILERS, AND A HASH-TABLE SHOULD BE USED FOR ANY REAL APPLICATIONS (SEE REFERENCE [1]).

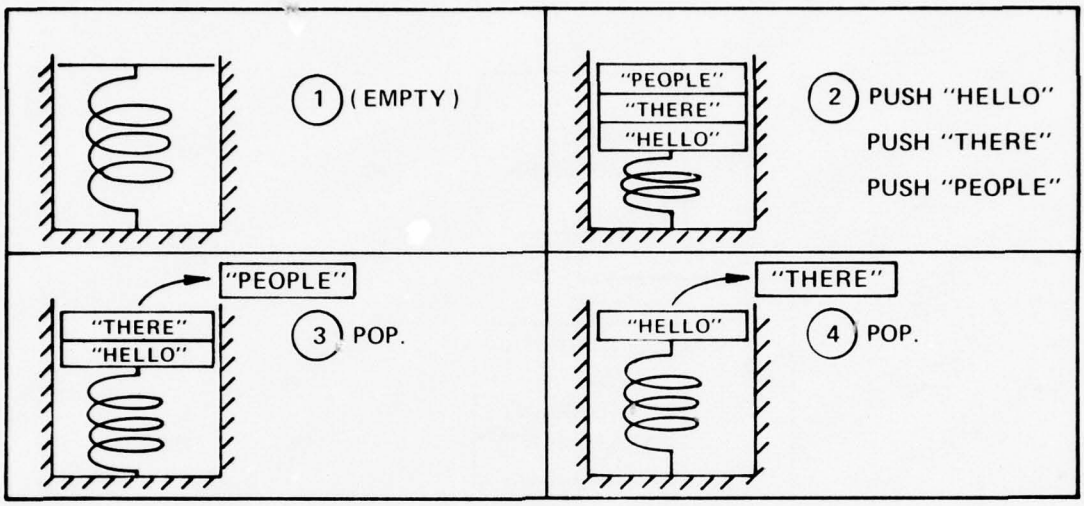
AFTER THE READER UNDERSTANDS THE PROGRAM OPERATION. AN INTERESTING EXERCISE CAN BE TO EXPAND THE PROGRAM TO INCLUDE FUNCTIONS, SUBSCRIPTED VARIABLES, AND EVEN CONDITIONAL STATEMENTS. ANOTHER INTERESTING EXERCISE IS TO WRITE A HIGH-LEVEL LANGUAGE FOR PROGRAMMABLE CALCULATORS. THIS HIGH-LEVEL LANGUAGE COULD BE CONVERTED INTO A SERIES OF KEYSTOKES USING RPN LOGIC BY A ROUTINE SIMILAR TO SUBROUTINE POLISH.

THIS PROBLEM WAS SELECTED BECAUSE IT ILLUSTRATES THE EASE WITH WHICH EVEN A RELATIVELY COMPLICATED STRING-MANIPULATION PROBLEM CAN BE HANDLED USING THE CHARACTER STRING PROCESSING LIBRARY. WITHOUT THE LIBRARY, THIS PROGRAM OF LESS THAN 220 STATEMENTS WOULD PROBABLY REQUIRE IN EXCESS OF 2000 STATEMENTS.

FIGURE B-1

PUSH-DOWN STACK

OPERATION - LAST IN, FIRST OUT (LIFO)



IMPLEMENTATION - ARRAY WITH POINTER

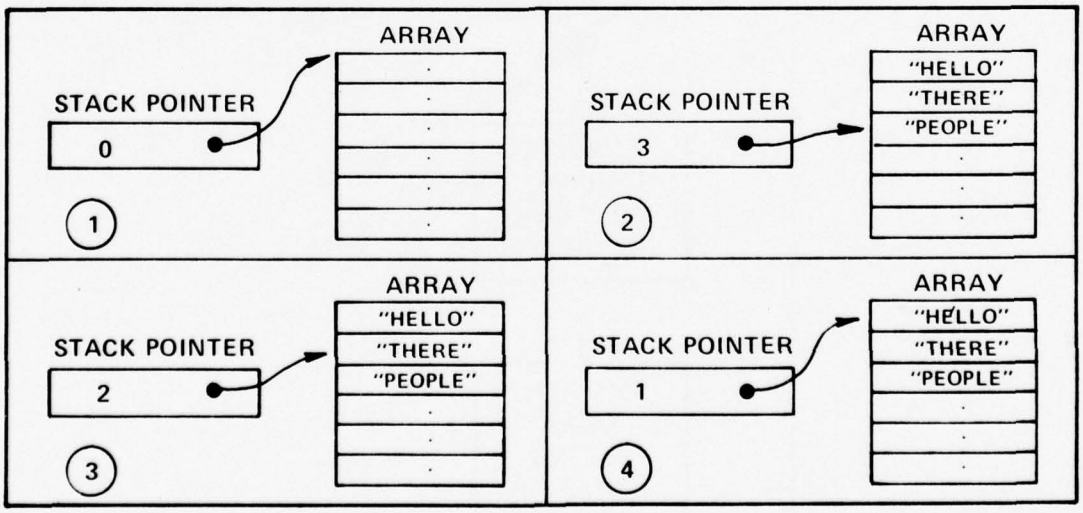


FIGURE B-2

PROGRAM XPRESN

SOLVE ALGEBRAIC ASSIGNMENT EXPRESSION

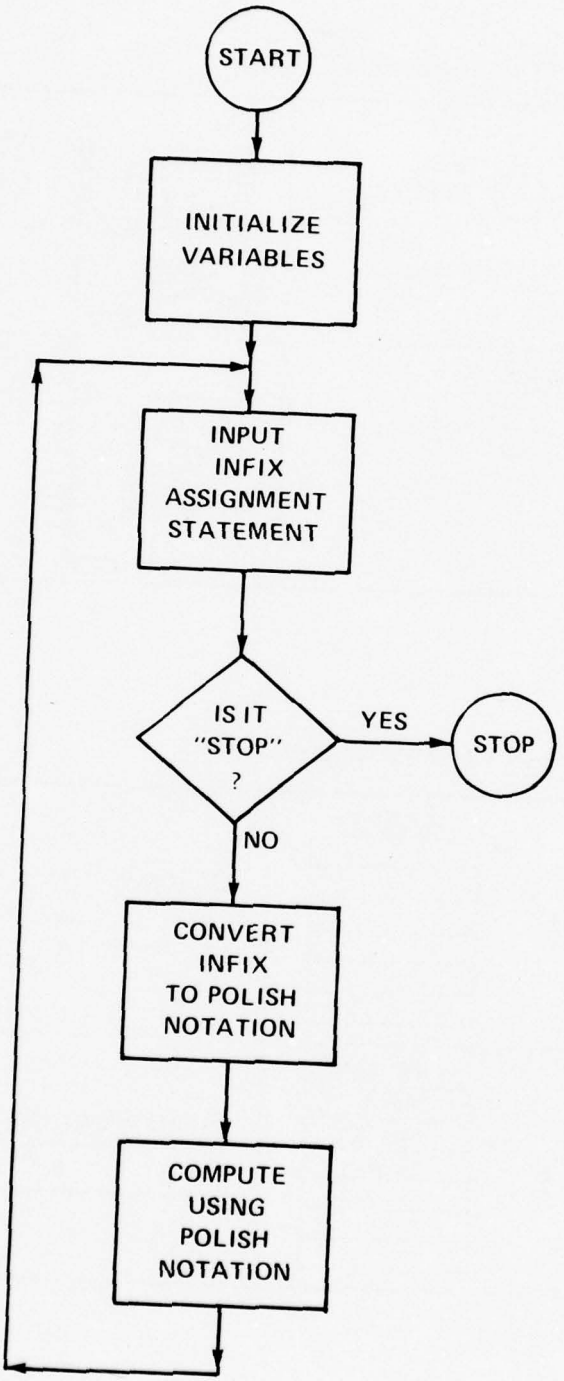
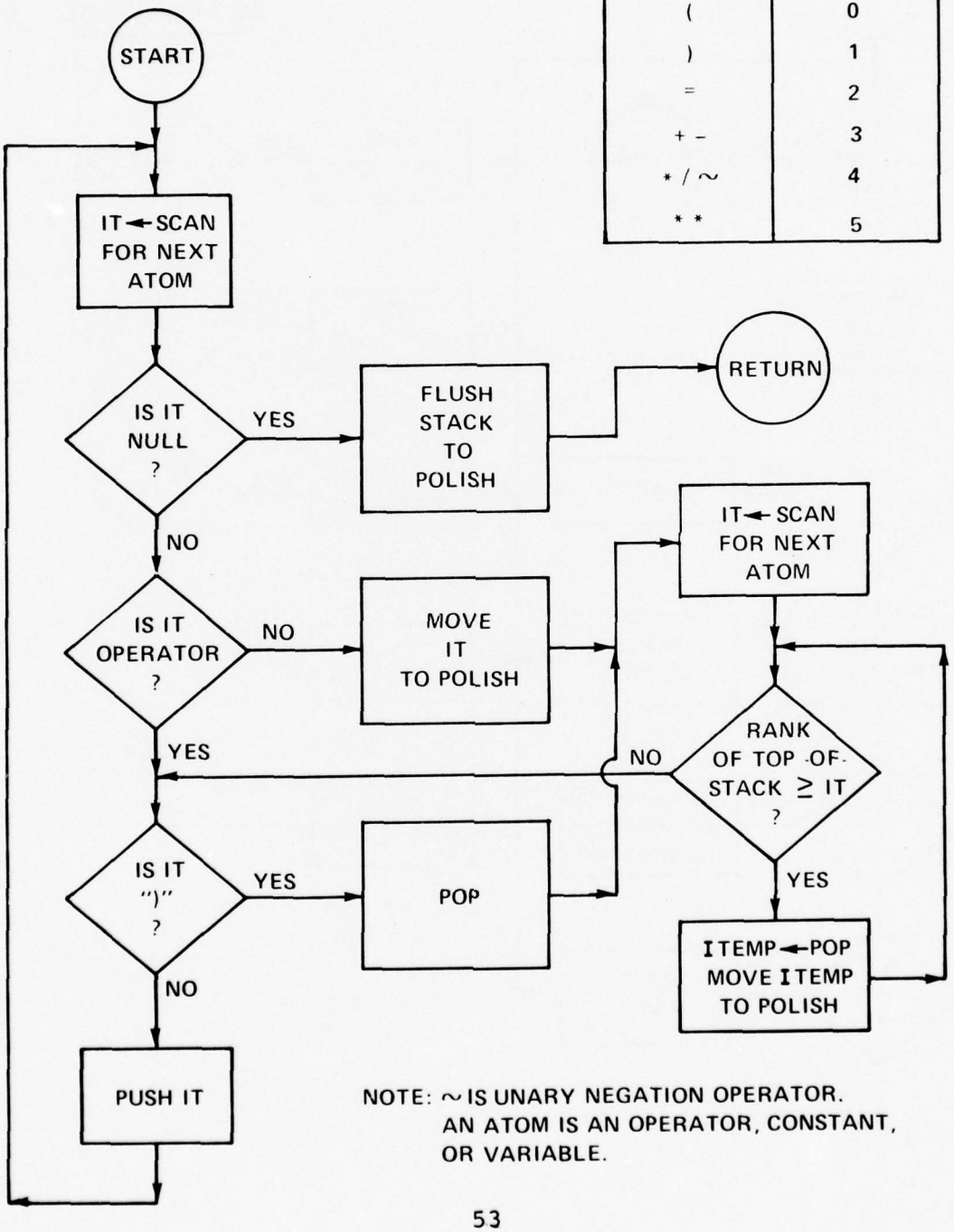


FIGURE B-3

SUBROUTINE POLISH

CONVERT INFIX TO POLISH NOTATION

OPERATOR HIERARCHY	
OPERATOR	RANK
(	0
)	1
=	2
+ -	3
* / ~	4
* *	5

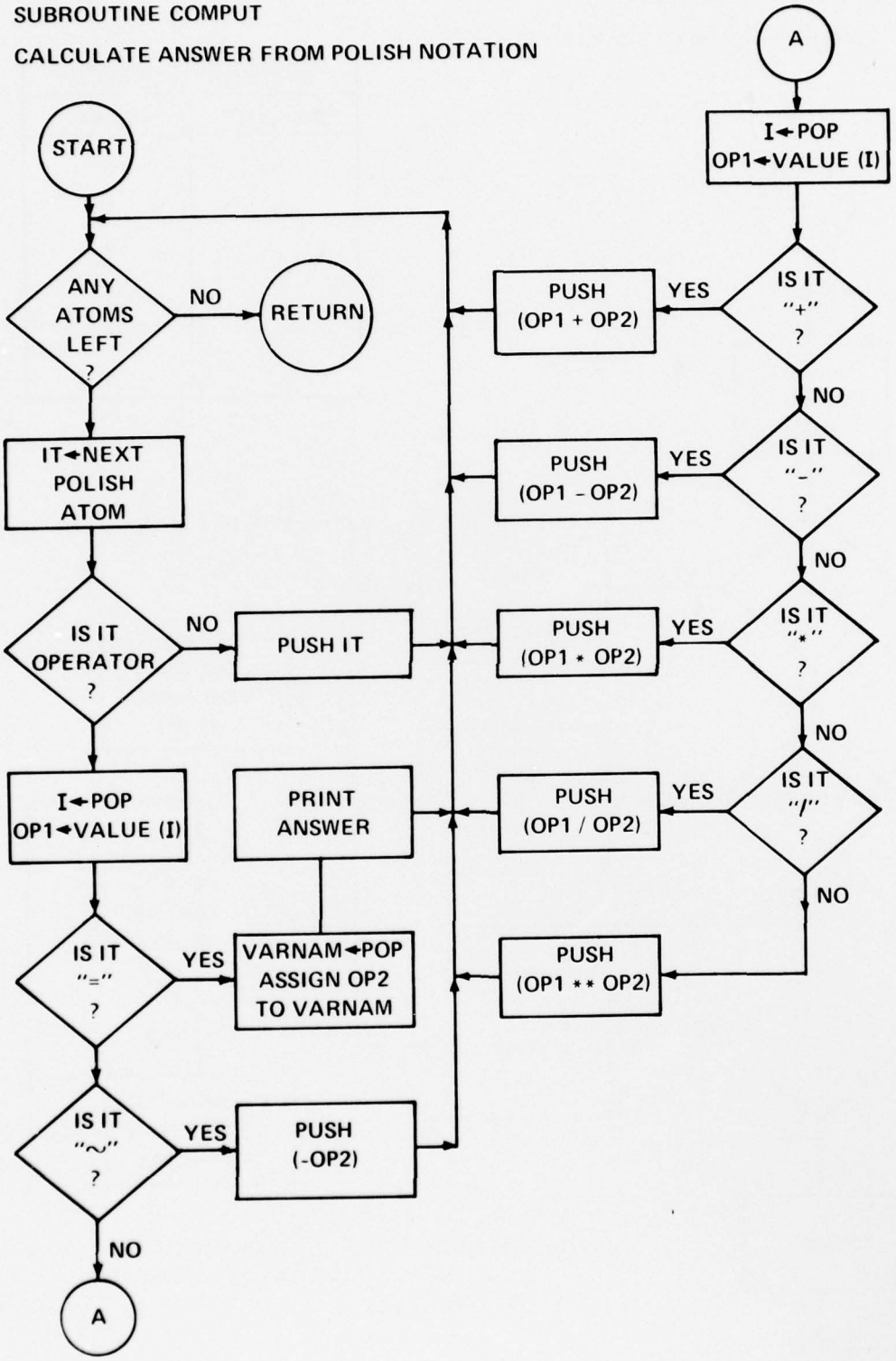


NOTE: ~ IS UNARY NEGATION OPERATOR.
AN ATOM IS AN OPERATOR, CONSTANT, OR VARIABLE.

FIGURE B-4

SUBROUTINE COMPUT

CALCULATE ANSWER FROM POLISH NOTATION



***** FORTRAN LISTING OF EXAMPLE PROGRAM 2 *****

```
      PROGRAM XPRESN(INPUT, TAPE5=INPUT, OUTPUT, TAPE6=OUTPUT)
C
C        INTERACTIVE ARITHMETIC EXPRESSION INTERPRETER WITH VARIABLES
C
      LOGICAL LEXIC, LASTOP
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LOLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
C        INITIALIZE COMMON VARIABLES.
      DATA LRANK / 0, 1, 2, 3, 3, 4, 4, 4, 5 /
      CALL KINIT(0)
      CALL KONLST(LOPER, "(", ")", "=", "+", "-", "*", "/",
     X "_", "**" )
      CALL KONLST(LOLIM, " ", "(", ")", "=", "+", "-", "*", "/",
     X "_", "**" )
      IBLK=KON(" ")
      INSTR=0
      DO 10 I=1,100
      LVAR(I)=0
      VARVAL(I)=0.0
   10 CONTINUE
      NVAR=
C
   11 CONTINUE
C        RESET COUNTERS IN PREPARATION FOR NEW EXPRESSION.
      IP=0
      LASTOP=.TRUE.
C        READ EXPRESSION.
      CALL KREPL(INSTR,KIN(5,-1))
      CALL KOUT(6,INSTR,1)
      IF(LEXIC(INSTR,"EQ",KON("STOP",-1)))STOP
C        CONVERT INFIX TO POLISH NOTATION.
      CALL POLISH
C        USE POLISH LIST TO COMPUTE AND PRINT RESULT.
      CALL COMPUT
      GO TO 11
      END
```

```
      SUBROUTINE POLISH
C
C        CONVERT EXPRESSION FROM INFIX NOTATION TO POLISH NOTATION
C
      LOGICAL LEXIC, LASTOP, LOP
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LOLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
      IPS=0
   51 CONTINUE
C        SCAN REMAINDER OF STRING, FIND NEXT ATOM,IT.
      CALL SCAN(IT,LOP)
C        IS IT AN OPERATOR...
      IF(LOP)GO TO 201
C        NO. ITS AN OPERAND IF THIS PATH...
C        MOVE IT TO POLISH LIST.
      IP=IP+1
      LPOL(IP)=IT
  111 CONTINUE
C        GET NEXT ATOM. IT.
      CALL SCAN(IT,LOP)
C        CHECK FOR IT=NULL MEANING END-OF-EXPRESSION.
      IF(IT.EQ.0)GO TO 251
  121 CONTINUE
C        COMPARE RANK OF OPERATOR STACK TOP WITH IT.
      IF(IRANK(ISEE(NS)).LT.IRANK(IT))GO TO 201
C        MOVE HIGHER-RANKING OPERATOR FROM STACK TO POLISH LIST.
      IP=IP+1
      LPOL(IP)=IPOP(NS)
      GO TO 121
  201 CONTINUE
C        COME HERE WHEN IT IS AN OPERATOR.  CHECK FOR IT=NULL=END
      IF(IT.EQ.0)GO TO 251
C        CHECK FOR SPECIAL CASE = RIGHT PARENTHESIS.
      IF(LEXIC(IT,"NE",LOPER(2)))GO TO 301
  221 CONTINUE
C        COME HERE WHEN ) ENCOUNTERED.
      JUNK=IPOP(NS)
      GO TO 111
  251 CONTINUE
C        COME HERE ON END-OF-LINE.  FLUSH OPERATOR STACK TO POLISH.
      JUNK=ISEE(NS)
  261 CONTINUE
      IF(NS.LT.1)RETURN
      IP=IP+1
      LPOL(IP)=IPOP(NS)
      GO TO 261
  301 CONTINUE
C          COME HERE WHEN ANY OPERATOR EXCEPT ")" IS ENCOUNTERED.
      CALL IPUSH(IT)
      GO TO 51
      END
```

```
      SUBROUTINE SCAN(IT,LOP)
C
C        SCAN REMAINING EXPRESSION FOR NEXT ATOM.
C         AN ATOM MAY BE AN OPERATOR, A VARIABLE, OR A NUMBER.
C
      LOGICAL LEXIC, LASTOP,LOP
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LDLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
C
      IF(INSTR.EQ.0)GO TO 151
C        LOCATE COLUMN WITH 1ST NON-BLANK CHARACTER.
      IINB=MATCH(INSTR,"FIRSTNOT",IBLK)
      IF(IINB.EQ.0)GO TO 151
C         REMOVE ANY LEADING BLANKS.
      IF(IINB.GT.1)CALL KREPL(INSTR,KSUB(INSTR,IINB))
C        FIND COLUMN WITH DELIMITER TERMINATING ATOM.
      ICDLIM=MATCH(INSTR,"FIRST",LDLIM,8,IMIT)
      IF(ICDLIM.EQ.1)GO TO 201
C        COME HERE IF IT IS AN OPERAND.
      IT=KSUB(INSTR,1,ICDLIM-1)
C        DISCARD PORTION OF EXPRESSION WERE DONE WITH.
      CALL KREPL(INSTR,KSUB(INSTR,ICDLIM))
      GO TO 501
  151 CONTINUE
C        COME HERE ON END-OF-STRING.
      IT=0
      LOP=.TRUE.
      GO TO 601
  201 CONTINUE
C        COME HERE IF IT IS AN OPERATOR.
      IT=KSUB(INSTR,1,1)
      IF(LEXIC(IT,"EQ",LOPER(6)))GO TO 251
      IF(LEXIC(IT,"NE",LOPER(5)))GO TO 401
C        COME HERE FOR - SIGN.  SEE IF IT IS UNARY OPERATOR _.
      IF(LASTOP)IT=LOPER(8)
      GO TO 401
  251 CONTINUE
C        CHECK FOR POSSIBLE ** OPERATOR INSTEAD OF JUST *.
      IF(LEXIC(KSUB(INSTR,2,1),"EQ",LOPER(6)))IT=LOPER(9)
  401 CONTINUE
      LASTOP=.TRUE.
      J=2
      IF(LEXIC(IT,"EQ",LOPER(9)))J=3
C        DISCARD PORTION OF EXPRESSION ALREADY DONE.
      CALL KREPL(INSTR,KSUB(INSTR,J))
      LOP=.TRUE.
      GO TO 601
  501 CONTINUE
      LOP=.FALSE.
      LASTOP=.FALSE.
  601 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE IPUSH(ITEM)
C
C        PUSH ITEM ONTO PUSH-DOWN (LIFO) STACK.
C
      LOGICAL LASTOP
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LDLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
      IPS=IPS+1
      LSTK(IPS)=ITEM
      RETUR
      END
```

```
      INTEGER FUNCTION IPOP(NS)
C
C        RETURN ITEM POPPED OF TOP OF PUSH-DOWN STACK.
C
      LOGICAL LASTOP
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LDLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
      IPOP=
      NS=IPS
      IF(IPS.LT.1)RETURN
      IPOP=LSTK(IPS)
      IPS=IPS-1
      NS=IPS
      RETUR
      END
```

```
      INTEGER FUNCTION ISEE(NS)
C
C        RETURN CONTENTS OF TOP OF PUSH-DOWN STACK WITHOUT AFFECTING
C        THE CONTENT OF THE STACK.
C         NS IS A VARIABLE, RETURNED AS PRESENT STACK DEPTH.
C
      LOGICAL LASTOP
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LDLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
      ISEE=
      NS=IPS
      IF(IPS.LT.1)RETURN
      ISEE=LSTK(IPS)
      RETUR
      END
```

```
      INTEGER FUNCTION IRANK(ISTR)
C
C        COMPUTE RELATIVE RANK OF OPERATOR, ISTR, FROM GIVEN HERARCHY.
C
      LOGICAL LEXIC, LASTOP
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LDLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
      IRANK=-1
      IF(ISTR.EQ.0)RETUR
      DO 100 I=1,9
      IF(LEXIC(LOPER(I),"EQ",ISTR))GO TO 101
  100 CONTINUE
      STOP" RANK ERROR"
  101 CONTINUE
      IRANK=LRANK(I)
      RETUR
      END
```

```
      SUBROUTINE COMPUT
C
C        EXECUTE OPERATIONS INDICATED BY POLISH EXPRESSION.
C
      LOGICAL LEXIC
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LOLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
      IPS=0
      IMSG=
      CALL KREPL(IMSG,KON(" =",-1))
      ANS=0.
      I=0
  101 CONTINUE
C        NEXT ITEM FROM POLISH LIST.
      I=I+1
      IF(I.GT.IP)GO TO 501
C        IS IT AN OPERAND...
      IT=LPOL(I)
      DO 15  J=1,9
      IF(LEXIC(IT,"EQ",LOPER(J)))GO TO 201
  150 CONTINUE
C        NO. IT IS AN OPERAND IF THIS PATH.  PUSH ON STACK.
      CALL IPUSH(IT)
      GO TO 101
  201 CONTINUE
C        YES, IT IS OPERATOR IF THIS PATH.  GET LAST 2 OPERANDS FROM
C        TOP-OF-STACK, UNLESS UNARY - OPER. OR = SIGN SPECIAL CASE.
      OP2=VALUE(IPOP(NS))
      IF(J. EQ.3)GO TO 401
      IF(J.NE.8)OP1=VALUE(IPOP(NS))
C         PERFORM THE OPERATION.
      JM3=J-3
      GO TO (301,311,321,331,341,351)JM3
  301 CONTINUE
C        ADD.
      ANS=OP1 + OP2
      GO TO 401
  311 CONTINUE
C        SUBTRACT
      ANS=OP1 - OP2
      GO TO 401
  321 CONTINUE
C        MULTIPLY.
      ANS= OP1 * OP2
      GO TO 401
  331 CONTINUE
C        DIVIDE.
      ANS=OP1 / OP2
      GO TO 401
```

```
  341 CONTINUE
C        UNARY NEGATION.
      ANS=-OP2
      GO TO 401
  351 CONTINUE
C        EXPONENTIATION.
      ANS=OP1 ** OP2
      GO TO 401
  381 CONTINUE
C        ASSIGNMENT ( = ).
      NAMVAR=IPOP(NS)
      DO 39  K=1,NVAR
      IF(LEXIC(NAMVAR,"END",LVAR(K)))GO TO 391
  390 CONTINUE
C        VARIABLE NOT FOUND, PUT IT IN TABLE.
      NVAR=NVAR+1
      LVAR(NVAR) = NAMVAR
      K=NVAR
  391 CONTINUE
      VARVAL(K)=OP2
      ANS=OP2
      CALL KREPL(IMSG,KAT(NAMVAR,IMSG))
      GO TO 501
  401 CONTINUE
C        PLACE RESULT OF OPERATION ON STACK.
      CALL IPUSH(KSR(ANS))
      GO TO 101
  501 CONTINUE
C        COME HERE TO PRINT ANSWER.
      CALL KOUT(6,KAT(IMSG,KSR(ANS)),1)
      RETURN
      END
```

```
      REAL FUNCTION VALUE(ITEM)
C
C        RETURN NUMERIC VALUE OF OPERAND ITEM.
C
C
      COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LOLIM(10),
     X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
     X NVAR
      VALUE=0.
      IF(ITEM.EQ.0)RETURN
C        1ST CHARACTER DETERMINES IF ITEM IS VARIABLE OR NUMBER.
      IC1=KSUB(ITEM,1,1)
      IF(LEXIC(IC1,"LT",KON("0")))GO TO 261
C        ITEM IS A LITERAL NUMBER IF THIS PATH.
      VALUE=RLVAL(ITEM)
      RETURN
  251 CONTINUE
C        COME HERE IF ITEM = VARIABLE.  FIND IT IN TABLE.
      DO 25  K=1,NVAR
      IF(LEXIC(ITEM,"EQ",LVAR(K)))GO TO 26
  25  CONTINUE
C        VARIABLE UNDEFINED IF THIS PATH, ASSUME = 0.
      CALL KOUT(6,KAT(KON(" UNDEFINED",-1),ITEM,KON(" SET TO 0.")))
      RETURN
  261 CONTINUE
C        RETURN VALUE ASSOCIATED WITH VARIABLE.
      VALUE = VARVAL(K)
      RETURN
      END
```

SAMPLE OUTPUT FROM PROGRAM XPRESN RUN

```
     Z= 11
Z = 11.
   ZFUNCTION = 1+(1/(2+(1/(6*Z+21))))
ZFUNCTION = 1.49714285714
 A =   1
A = 1.
B=10.000
B = 10.
 C =11+5.5+3.5 -1
C = 21.
  ROOT= (-B + (B*B - 4*A*C))/(2*A)
ROOT = 3.
   T = 175
T = 175.
  GR= 7.6E13
GR = 7.600000E+13
PR=.2256
PR = .2256
  NUSUBL = .0246 * ((PR**1.17)/(1+.049*PR**.67)*GR)**0.4
NUSUBL = 4340.2204221
 FINAL   =  NUSUBL/2  +  ZFUNCTION
FINAL = 2171.6073539l
JOE = RALPH + 2.44
UNDEFINED RALPH SET TO 0.
JOE = 2.44
STOP
```

APPENDIX C

ALPHABETICAL LISTING OF KSPLIB ROUTINES AND LENGTHS

(SINCE THE CHARACTER STRING PROCESSING ROUTINES ARE AVAILABLE ON A LIBRARY, ONLY PROGRAMS ACTUALLY NEEDED ARE LOADED)

	NAME	OCTAL LEN.	REFERENCES THESE ROUTINES...
	BIT	43	
	CENSUS	11	
	CHFILL	116	CHXFR
	CHXFR	115	
	DSGET	371	BIT CENSUS IFETS VFETP VIO
	IFETS	25	
*	INTVAL	151	KAT KON KOUT KASA NFIXER
*	KAT	156	CHXFR DSGET KSAS KSMAKE LOCFP
*	KDC	110	CHXFR DSGET KSMAKE
*	KEC	66	CHXFR KSAS
*	KIN	252	CHXFR DSGET KRACTR KSMAKE
*	KINIT	200	VFETC8 VFETP VIO
*	KINTEK	137	AINST CHXFR DSGET KRACTR KSMAKE
*	KON	260	CHFILL CHXFR DSGET KRACTR KSMAKE
*	KONLST	61	LOCFP
*	KOUT	215	KSAS
*	KOUTEK	103	KSAS
	KRACTR	61	
*	KREPL	37	KRLS
*	KRLS	120	KSAS LOCFP
*	KRLSL	70	KRLS
	KSAS	222	IFETS VFETP VIO
	KSMAKE	114	
*	KSR	413	CHXFR DSGET KRACTR KSMAKE
*	KSUB	135	CHXFR DSGET KSAS KSMAKE
*	LEN	42	KSAS
*	LEXIC	353	CHXFR KRACTR KSAS
	LOCFP	26	KON
*	MATCH	526	CHXFR KRACTR KSAS
	NFIXER	16	NFLTER
	NFLTER	127	
*	RLVAL	146	KAT KON KOUT KSAS NFLTER
	VFETC8	65	
	VFETP	67	
	VIO	60	

*NOTE: ROUTINES ACCESSABLE BY USER.
LENGTH OF ALL ROUTINES IS SUBJECT TO CHANGE AS MODIFICATIONS ARE MADE TO LIBRARY OR OPERATING SYSTEM.

REFERENCES

[1] KNUTH, DONALD E., THE ART OF COMPUTER PROGRAMMING, VOLUME III, ADDISON-WESLEY, 1973.

[2] GRIES, DAVID, COMPILER CONSTRUCTION FOR DIGITAL COMPUTERS, JOHN WILEY AND SONS, 1971.

[3] CONTROL DATA CORPORATION, SCOPE REFERENCE MANUAL, 60307200.

[4] CONTROL DATA CORPORATION, CYBER RECORD MANAGER VERSION 1, 60307300.

[5] CONTROL DATA CORPORATION, FORTRAN EXTENDED VERSION 4 REFERENCE MANUAL, 60305600.

[6] ACM COMMITTEE X3J3, FORTREV, ACM, P.O. BOX 12105, CHURCH ST. STATION, N.Y., N.Y. 10249.

[7] PETERSON, W. WESLEY, INTRODUCTION TO PROGRAMMING LANGUAGES, PRENTICE-HALL, 1974.

[8] GRISWOLD, R.E., ET AL, THE SNOBOL 4 PROGRAMMING LANGUAGE, PRENTICE-HALL, 1971.

ALPHABETICAL QUICK-REFERENCE INDEX TO CHARACTER-STRING ROUTINES

CALLING FORMAT(S)	PAGE
<INTEGER VAL.> = INTVAL(<STRING REP.>) <INTEGER VAL.> = INTVAL(<STRING REP.>,<IF ERROR>)	6,26
<STR.REP.> = KAT(<STR.REP.>,<STR.REP.>[,<STR.REP.>...])	3,14
<STR.REP.> = KOC(<ACTUAL-STRING>,<COL.1>,<LENGTH>)	41
CALL KEC(<STR.REP.>,<DEST.>,<COL.1>)	42
<STR.REP.> = KIN(<LEN>) <STR.REP.> = KIN(<LEN>,<LENGTH>) <STR.REP.> = KIN(<LEN>,-<PADDING>)	4,21
CALL KINIT(<BUF.SIZE>)	7,34,38
<STR.REP.> = KINTEK(<LENGTH>)	24
<STR.REP.> = KON(<LITERAL STR.>) <STR.REP.> = KON(<LITERAL STR.>,<LENGTH>) <STR.REP.> = KON(<LITERAL STR.>,-<PADDING>)	3,11
CALL KONLST(<INT.ARRAY>,<LIT.STR.>,[<LIT.STR.>...])	13
CALL KOUT(<LEN>,<STR.REP.>) CALL KOUT(<LEN>,<STR.REP.>,<CAR.CNTRL.>)	4,23
CALL KOUTEK(<STR.REP.>) CALL KOUTEK(<STR.REP.>,<CAR.CNTRL.>)	25
CALL KREPL(<STR.REP.>,<STR.REP.>)	33
CALL KRLS(<STR.REP.>[,<STR.REP.>...])	39
CALL KRLSL(<STR.REP.ARRAY>,<NO.RLS.>)	40
<STR.REP.> = KSR(<VALUE>) <STR.REP.> = KSR(<VALUE>,<FTN.FMT>)	31
<STR.REP.> = KSUB(<STR.REP.>,<START.COL.>) <STR.REP.> = KSUB(<STR.REP.>,<START.COL.>,<NO.CHAR.>)	3,15
<INT.VAL.> = LEN(<STR.REP.>)	5,16
<.T./.F.> = LEXIC(<STR.REP.>,<KEY>,<STR.REP.>)	5,17
<COL.NO.> = MATCH(<STR.REP.>,<KEY>,<PAT.S.R.>) <COL.NO.> = MATCH(<STR.REP.>,<KEY>,<PAT.LIST S.R.>,<NO.PAT.>,<I MATCH>)	5,19
<REAL VAL.> = RLVAL(<STR.REP.>) <REAL VAL.> = RLVAL(<STR.REP.>,<IF ERROR>)	29

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: MISD-UM-77-2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): KSPLIB User Manual. CHARACTER STRING PROCESSING LIBRARY ROUTINES		5. TYPE OF REPORT & PERIOD COVERED: FINAL rept. 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): BRUCE D. CARBREY		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Management Information Systems Directorate, ARRADCOM, Dover, N.J. 07801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE: April 1977 13. NUMBER OF PAGES: 70
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CHARACTER	SUBSTRING	COMPILERS
STRING	PATTERN MATCHING	INTERPRETERS
TEXT	EDITOR	
CONCATENATION		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes a set of general-purpose, FORTRAN-callable functions and subroutines for manipulating character-strings. Character-strings of arbitrary length can be treated as simple (non-dimensioned) variables. Functions are provided for string definition, string concatenation, substring extraction, string comparison, pattern matching and searching, numeric-string/string-numeric conversions, string input/output, and other tasks. The routines described are moderately system-dependent, and are currently available

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 191

as an object program library on the CDC 6500/6600 computers at the ARRADCOM Dover, New Jersey site.